НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів»

Виконав:
студент IV курсу, групи ТВ-61
Мартиненко Олександр Петрович _____

Керівник:
проф. каф. АПЕПС, д.т.н., доцент Шушура О.М. _____

Рецензент:
зав кафедри Інформаційних систем та технологій
Державного університету телекомунікацій,
д.т.н., професор Сторчак К.П. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Мартиненку Олександру Петровичу
(прізвище, ім’я, по батькові)

1. Тема роботи ___технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів
керівник роботи проф. каф. АПЕПС, д.т.н., доцент Шушура О.М.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р.
№ **1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи алгоритми оптимізації гіперпараметрів моделей машинного навчання; набір даних MNIST для тестування класифікатора зображень на основі методу опорних векторів
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)___Аналіз задачі мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів; огляд існуючих алгоритмів та інформаційних технологій оптимізації значень гіперпараметрів моделей машинного навчання; вибір засобів розробки програмного продукту; проектування сервісу пошуку оптимальних значень гіперпараметрів, який надає прикладний програмний інтерфейс для взаємодії з клієнтськими додатками та графічний веб-інтерфейс для користувача; розробка програмного забезпечення та його тестування на наборі прикладів; підготовка опису програмного продукту для користувача.

5. Перелік ілюстративного матеріалу

Актуальність, Алгоритми оптимізації, Існуючі системи, Постановка задачі, Засоби розробки, Взаємодія з сервісом, API, Веб-інтерфейс, Тестування GridSearch, Тестування BayesianOpt, Тестування OnePlusOne, Тестування: класифікація зображень, Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	10.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019 – 27.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020 – 06.03.2020	
4.	Розробка структур окремих підсистем	10.03.2020 – 10.04.2020	
5.	Програмна реалізація системи	13.04.2020 – 15.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020 – 05.06.2020	
7.	**Захист програмного продукту**	09.06.2020	
8.	**Передзахист**	09.06.2020	
9.	Захист	16.06.2020	

Студент _____ О.П. Мартиненко
(підпис) (прізвище та ініціали,)

Керівник роботи _____ О.М. Шушура
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

За обсягом робота займає 61 сторінку, містить 36 рисунків, 1 таблицю, 4 додатки та посилається на 20 бібліографічних найменувань. Метою роботи є розробка програмного сервісу для автоматизації процесу мінімізації похибок моделей машинного навчання шляхом пошуку оптимальних гіперпараметрів. У роботі розглянуто використання бібліотек, програмних систем на основі хмарних технологій та платформ, що дозволяють мінімізувати похибки моделей машинного навчання шляхом оптимізації гіперпараметрів. Задля забезпечення безпеки даних та збереження незалежності від технологій користувача реалізовано сервіс, що розгортається на стороні користувача та надає прикладний програмний інтерфейс для взаємодії з користувацькими моделями, а також веб-інтерфейс для контролю процесу оптимізації. Роботу апробовано на конференції «VIII Всеукраїнська науково-практична конференція з автоматичного управління».

Ключові слова: оптимізація гіперпараметрів, машинне навчання, наука про дані, сервіс, прикладний програмний інтерфейс, Python.

ABSTRACT

The thesis consists of 61 pages, contains 36 pictures, 1 table, 4 attachments, refers to 20 references. The aim of the thesis is to implement service to facilitate minimization of ML-model errors via hyperparameter optimization. Usage of libraries, cloud-based program systems and platforms for hyperparameters optimization are described. For the sake of data security while keeping technology-independency the self-hosted service was developed. It provides API for integration into customer’s pipeline and web-interface for visualization. Taken decisions discussed in the conference «VIII Всеукраїнська науково-практична конференція з автоматичного управління».

Keywords: hyperparameter optimization, machine learning, data science, service, API, Python.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DBNs (англ. Deep Belief Networks, також глибинна мережа переконань) – родина моделей машинного навчання, що є різновидом глибинних штучних нейронних мереж.

TPE (англ. Tree of Parzen Estimators) – родина алгоритмів оптимізації гіперпараметрів.

Jupyter Notebook – інтерактивне середовище розробки, що поєднує інструменти розробки, документування та виконання коду, а також візуалізації результатів.

ВСТУП

Дослідження в області машинного навчання фокусуються на розробці методів, які здатні знаходити певні закономірності у даному наборі даних. Такі закономірності включають, але не обмежуються, структурою даних (кластеризація) або здатністю передбачати певні цільові значення на основі заданих характеристик, що можуть бути дискретними (класифікація) або неперервними (регресія). Існує велика різноманітність методів машинного навчання: біологічно інспіровані нейронні мережі, ансамблеві моделі, тощо.

Загальною рисою цих методів є те, що вони параметризовані набором гіперпараметрів, який повинен бути встановлений користувачем належним чином для отримання максимальної користі від обраного методу. Гіперпараметри використовуються для налаштування різних аспектів алгоритму навчання та можуть мати різний вплив на отриману модель та її ефективність.

Зазвичай значення гіперпараметрів обирають ітеративно, тренуючи модель машинного навчання з різними значеннями гіперпараметрів та порівнюючи отримані результати. Процес оптимізації гіперпараметрів може займати досить тривалий час відносно загального часу, що витрачається на створення та впровадження моделей машинного навчання на практиці, тому розробка інструментів для полегшення цієї задачі може принести користь багатьом розробникам та спеціалістам зі сфери науки про дані.

1 ПОСТАНОВКА ЗАДАЧІ

Метою даної роботи є розробка програмного сервісу для автоматизації процесу мінімізації похибок моделей машинного навчання шляхом пошуку оптимальних гіперпараметрів. Для досягнення цієї мети необхідно розв’язати наступні задачі:

1. Зробити огляд існуючих алгоритмів пошуку отптимальних гіперпараметрів у багатовимірному просторі та обрати алгоритми, що слід реалізувати у власному сервісі.
2. Провести огляд існуючих програмних рішень, що надають можливості оптимізації гіперпараметрів моделей машинного навчання. Описати переваги та недоліки різних програмних рішень та обгрунтувати актуальність створення нового програмного засобу для вирішення поставленої задачі.
3. Вибрати засоби розробки сервісу, як то: мову програмування, інструменти що полегшать розробку, засоби поширення розробленого сервісу. Навести альтернативи, обгрунтувати вибір.
4. Прийняти архітектурні рішення щодо структури проекту, визначити інтерфейс взаємодії користувача з сервісом. Описати імплементований функціонал.
5. Провести мануальне тестування сервісу, надати користувачеві тестові приклади скриптів для наочної демонстрації можливостей системи та як засіб перевірки правильності розгортання сервісу.
6. Надати детальний опис інтерфейсу взаємодії користувача з системою. Навести приклади запитів до систем та отриманих відповідей.

Розроблена система має мати можливість запуску на стороні користувача, бути незалежною від мов програмування, фреймворків, хмарних сервісів чи платформ, що використовуються користувачем для тренування моделей машинного навчання. Сервіс має запускатися у контеризованому вигляді засобами інструменту docker, надавати як прикладний програмний інтерфейс для взаємодії з програмними застосунками користувача, так і графічний веб-інтерфейс для візуального контролю

прогресу оптимізації. Допустимий розмір docker-образу сервісу не має перевищувати 400 мегабайт, вимоги оперативної пам'яті бути у межах 200 мегабайт при проведені оптимізації одної моделі машинного навчання популярними алгоритмами оптимізації. Сервіс має надавати користувачам можливість реалізовувати власні алгоритми оптимізації задля розширення функціоналу.

2 ХАРАКТЕРИСТИКА ЗАДАЧІ МІНІМІЗАЦІЇ ПОХИБОК МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

Мета багатьох завдань машинного навчання може бути визначена як тренування моделі, що мінімізує деяку заздалегідь задану функцію витрат L за даними тестових даних X_test. Загальні функції витрат включають середню квадратичну похибку та долю помилок. Модель, що побудована алгоритмом машинного навчання A на навчальному наборі X_train є, як правило, результатом вирішення якоїсь проблеми оптимізації. Алгоритм навчання A може сам параметризуватися набором гіперпараметрів λ. Наприклад, проблема навчання A параметризована постійною регуляризації C і пропускною здатністю σ, тоді λ = [C, σ].

Метою оптимізації гіперпараметрів є пошук набору гіперпараметрів, які дають оптимальну модель, що мінімізує функцію витрат.

Цільова функція даної задачі оптимізації приймає кортеж гіперпараметрів λ і повертає пов'язане з ними значення функції витрат. Дані множини X_train і X_test задано і вибрано алгоритм навчання A і функцію витрат L. Залежно від завдання, X_train і X_test можуть бути розмічені та/або рівні один одному. У навчанні з вчителем набір даних часто розбивається на X_train і X_test з використанням методів відкладеного контролю або перехресної перевірки [1].

Кожне оцінювання цільової функції вимагає оцінки працездатності моделі, навченої з гіперпараметрами λ. Залежно від наявних обчислювальних ресурсів, характеру алгоритму навчання A і розміру задачі (X_train, X_test) кожна оцінка може зайняти значний час. Час тренування порядка хвилин вважають швидкими, оскільки дні і навіть тижні не є нечуваними. Час оцінювання збільшується, коли застосовуються процедури, які навчають декілька моделей; наприклад, щоб надійно оцінити ефективність узагальнення. Це призводить до зростаючої потреби в ефективності методу оптимізації гіперпараметрів, що вимагає мінімальної кількості обрахунків цільової функцій. Крім того, час, необхідний для підготовки та тестування моделей, може залежати від вибору гіперпараметрів. Деякі гіперпараметри мають

очевидний вплив на час тренування та/або тестування моделі, наприклад, архітектури нейронних мереж та розмір ансамблів. Вплив гіперпараметрів також може бути неоднозначним, наприклад регуляризація іноді може суттєво вплинути на час навчання для методу опорних векторів [2].

Цільова функція часто має стохастичний компонент, який може бути внесеним різними компонентами процесу машинного навчання, наприклад, завдяки притаманній алгоритму навчання випадковості (ініціалізація нейронної мережі, перекомпонування в ансамблевих підходах, тощо) або завдяки кінцевим ефектам вибірки при оцінці ефективності узагальнення. Цю стохастичність іноді можна вирішити за допомогою технік машинного навчання, але, на жаль, такі рішення, як правило, значно збільшують необхідний час оцінки цільової функції, обмежуючи їх корисність у деяких випадках. Ця притаманна стохастичність прямо означає, що отриманий найкращий емпіричний набір гіперпараметрів після заданої кількості оцінок не обов'язково є справжнім оптимальним набором. Ця проблема частково вирішується тим, що багато методів пошуку розроблені для дослідження багатьох кортежів гіперпараметрів, близьких до найкращого емпіричного. Якщо область пошуку навколо емпіричного оптимуму має щільну кількість оцінених точок, ми можемо визначити, чи найкращий емпіричний результат не був випадковою похибкою на етапі пост-обробки, наприклад, передбачаючи неперервність або гладкість за Ліпшицом [3].

Пошук оптимальних гіперпараметрів дуже часто проводиться вручну, за допомогою емпіричних правил, спираючись на досвід фахівця або шляхом тестування різних наборів гіперпараметрів на заздалегідь заданій сітці. Ці підходи залишають бажати кращого з точки зору відтворюваності результатів і недоцільні, коли кількість гіперпараметрів велика. Через ці вади, ідея автоматизації пошуку оптимальних гіперпараметрів привертає все більше уваги спеціалістів у області машинного навчання. Автоматизовані підходи вже демонструють результати, що перевершують результати ручного пошуку експертами у кількох областях. Для пошуку гіперпараметрів використовується широке різноманіття методів оптимізації, включаючи particle swarm оптимізацію, генетичні алгоритми, coupled simulated

annealing та racing алгоритми. Випадковий пошук у просторі розв’язків лише відносно нещодавно був встановлений за базовий рівень. Баєсовим та спорідненим методам оптимізації на основі послідовних моделей з використанням варіантів очікуваного критерію вдосконалення приділяють багато уваги в даний час, завдяки їх ефективності з точки зору цільової функцій. Випускаються програмні пакети, які реалізують різні спеціалізовані методи оптимізації для пошук гіперпараметрів. Такі пакети зазвичай призначені для використання у зв’язці з бібліотеками, які надають алгоритми машинного навчання. Більшість цих пакетів зосереджені на байєсівських методах, хоча метагевристичні оптимізаційні підходи також пропонуються. Посилений розвиток таких пакетів свідчить про зростаючий інтерес до автоматизованого пошуку гіперпараметрів [4].

3 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ РОЗРАХУНКУ ГІПЕРПАРАМЕТРІВ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

Попередні дослідження демонструють, що проблема оптимізації гіперпараметрів у великих моделях, зокрема багатошарових штучних нейронних мережах, є прямою перешкодою для прогресу у областях використання даних моделей. У певних роботах показники якості проблем класифікації зображень збільшуються здебільш завдяки концентрації зусиль на оптимізації гіперпараметрів у простих алгоритмах, а не за допомогою винаходження нових моделей чи стратегій машинного навчання. Неправильно було б зробити висновок, що докладання зусиль на виокремлення більш складних характеристик з даних є марним. Натомість оптимізацію гіперпараметрів слід розглядати як формальну зовнішній цикл в процесі навчання.

Алгоритм машинного навчання, як функціонал від даних до класифікатора (беручи до прикладу проблему класифікації), включає вибір, скільки процесорних ресурсів потрібно витратити на дослідження гіперпараметрів і скільки ресурсів потрібно витратити на оцінку кожного набору гіперпараметрів (тобто на оптимізацію звичайних параметрів). Результати досліджень дозволяють припустити, що в умовах сучасного стану доступності апаратних засобів, таких як великі комп'ютерні кластери та графічні процесори, оптимальне розподілення циклів процесора включає в себе більше вивчення гіперпараметрів, ніж це наводиться в класичній літературі по машинному навчанню [5].

Для пошуку значень гіперпараметрів, що мінімізують похибки моделей машинного навчання, використовують досить багато різноманітних алгоритмів. Реалізації цих алгоритмів дещо відрізняються від одного програмного рішення до іншого, так само як і наявність певних алгоритмів у певних інструментах. Тому слід розглянути кілька популярних методів оптимізації та навести огляд існуючих програмних технологій, щоб обрати перспективні алгоритми і техніки реалізації для створення власного сервісу.

3.1 Методи оптимізації гіперпараметрів

В якості базових математичних методів вирішення даної задачі наводять пошук за сіткою, що вичерпно розглядає всі комбінації гіперпараметрів, та випадковий пошук, що обирає наступний набір гіперпараметрів для розрахунку цільової функції випадково за заданим розподілом.

Хоча випадковий пошук і часто показує істотний результат, розробляються кращі алгоритми. Деякі моделі машинного навчання, такі як Deep Belief Networks (DBNs), stacked denoising autoencoders, згорткові мережі, а також класифікатори основані на складних методах виокремлення характеристик у вхідних даних, мають від десяти до, можливо, п’ятдесяти гіперпараметрів. Це залежить від того, як експериментатор параметризує модель та від кількості гіперпараметрів, які експериментатор фіксує певним значенням за замовчуванням. Труднощі налаштування цих моделей ускладнюють відтворення опублікованих результатів, заважають розширювати їх, і роблять дослідження таких методів занадто непередбачуваним.

Серед інших методів, для пошуку оптимальних параметрів використовують генетичні алгоритми. Генетичний алгоритм - це еволюційний алгоритм пошуку, який використовується для вирішення проблем оптимізації та моделювання шляхом послідовного вибору, комбінування та зміни параметрів за допомогою механізмів, що нагадують біологічну еволюцію. Генетичні алгоритми імітують процес природного відбору, тобто ті варіації, які показують гарний результат роботи у заданому середовищі, можуть вижити та перейти до наступного покоління. Кожне покоління складається з популяції індивідів, і кожен індивід являє собою точку в просторі пошуку гіперпараметрів. Кожна особа представлена у вигляді рядка значень (символів або чисел що є значеннями гіперпараметрів). Ця послідовність є аналогом хромосоми. Генетичний алгоритм починається з випадково згенерованої популяції хромосом. Потім проводиться процес рекомбінації на основі придатності кожної хромосоми (тобто, на основі результата, що дає цільова функція при даних гіперпараметрах). Батьківські генетичні матеріали рекомбінуються для генерування

дочірніх хромосом, з яких складається наступне покоління. Цей процес ітеративно повторюється [6].

Рекомбінація – це процес, за допомогою якого хромосоми, відібрані з вихідної популяції, рекомбінуються для формування членів популяції-наступника. Ідея полягає в імітації змішування генетичного матеріалу, що може відбуватися при розмноженні організмів. Оскільки відбір для рекомбінації є упередженим на користь вищої придатності, мета рекомбінації полягає в тому, що в результаті розвиватимуться більш придатні хромосоми. Є два основні компоненти рекомбінації: кросовер та мутація, вони зазвичай виконуються послідовно. Зміни внесені у процесі рекомбінації недетерміновані. Кожна зміна хромосом відбувається з певною вірогідністю, і точний результат кросоверу або мутації не є детермінованим.

Кросовер представляє змішування генетичного матеріалу з двох вибраних батьківських хромосом для отримання однієї або двох дочірніх хромосом. Після вибору двох батьківських хромосом для рекомбінації генерується випадкове число в інтервалі від нуля до одиниці з рівномірною ймовірністю та порівнюється із заздалегідь визначеною “швидкістю кроссовера”. Якщо випадкове число більше швидкості кросовер, кросовер не відбувається, і один або обидва батьки переходять без змін на наступний етап рекомбінації. Якщо швидкість кросоверу більша або дорівнює випадковому числу, то застосовується власне кросовер.

Один широко використовуваний варіант кросовера – це одноточковий кросовер. На довжині хромосоми обирається точка перехрестя (зазвичай вона обирається з рівномірного розподілу). Потім дочірня хромосома будується з символів першого батька, що йдуть перед точкою перехрещення, та символів другого з батьків, що йдуть після точки перехрестя. Існує безліч альтернативних форм роботи кросовера. Одноточковий кросовер узагальнюється до двох- та багатоточкових варіантів кроссовера, де кілька точок кроссовера вибирають по довжині хромосоми, а дочірні хромосоми будуються із значень генів двох батьків, змінюючись у кожній точці кроссовера.

Оператори мутації застосовують на окремі хромосоми, щоб змінити один або більше значень генів. Випадкове число в інтервалі [0, 1] формується з рівномірного

розподілу і порівнюється з заздалегідь визначеною "швидкістю мутації". Якщо випадкове число більше, ніж частота мутацій, то для цього гену не застосовується жодна мутація. Якщо коефіцієнт мутації більший або дорівнює випадковому числу, то значення гену змінюється на випадкове інше значення. Частота мутації зазвичай дуже мала (наприклад, 0.001).

Після рекомбінації отримані хромосоми передаються в наступну популяцію. Процеси відбору та рекомбінації потім повторюються до отримання повної популяції спадкоємців. Так генерується наступне покоління. Алгоритм проводить ітерації протягом декількох поколінь до досягнення відповідних критеріїв зупинки. До таких критеріїв можна віднести фіксовану кількість поколінь, які минули; зменьшення швидкості збіжності алгоритму, або знаходження рішення, що повністю задовольняє набір обмежень.

Існує кілька еволюційних схем, які можуть бути використані, залежно від того, наскільки хромосомам з вихідної популяції дозволяється без змін змінюватися в популяції-наступнику. Вони варіюються від повної заміни, коли всі члени покоління-наступника генеруються шляхом відбору та рекомбінації, до стаціонарного стану, де популяція-наступник створюється шляхом генерування по одній новій хромосомі в кожному поколінні та використання її для заміни найменьш придатного члена попередньої популяції.

Вибір еволюційної схеми є важливим аспектом проектування генетичних алгоритмів і залежатиме від характеру простору рішення, що шукається. Широко застосовувана схема - це заміна з елітаризмом. Це майже повна заміна попереднього покоління, за винятком того, що найкраща одна чи дві особи з вихідної популяції зберігаються в популяції спадкоємців. Ця схема запобігає втраті рішень найвищої відносної придатності при переході до наступного покоління через недетермінований процес відбору [7].

Також для пошуку оптимальних гіперпараметрів використовують байєсівську оптимізацію (часним випадком якої є моделювання функції гауссівськими процесами). Байєсівську оптимізацію відмінною від інших процедур робить те, що вона будує імовірнісну модель для простору гіперпараметрів, а потім використовує

цю модель для прийняття рішень про те, де в цьому просторі далі оцінити функцію, обчислюючи при цьому невизначеність у даній точці простору гіперпараметрів. Основна філософія полягає у використанні всієї інформації, наявної з попередніх оцінок цільової функції, а не просто у тому щоб покладатися на локальні градієнти.

Така оптимізація називається байєсівською, оскільки вона використовує знамениту «теорему Байєса», в якій говориться (дещо спрощуючи), що постеріорна ймовірність теорії (або гіпотези) M за даними спостережень E пропорційна ймовірність E за умови M, помножену на апріорну ймовірність M:

$$P(M|E) \propto P(E|M)P(M). \quad (3.1)$$

Визначимо x_i як і-тий набір гіперпараметрів, а $f(x_i)$ як значення цільові функції при xi. Оскільки ми накопичуємо спостереження $D_{1:t} = \{x_{1:t}, f(x_{1:t})\}$, пріорний розподіл поєднується з функцією ймовірності $P(D_{1:t} \mid f)$. Якщо наша попередня переконання полягає в тому, що цільова функція дуже гладка і безшумна, дані з великою дисперсією або коливаннями слід вважати менш вірогідними, ніж дані, які ледь відхиляються від середнього значення. Тепер можена комбінувати їх для отримання постеріорного розподілу:

$$P(f|D_{1:t}) \propto P(D_{1:t}|f)P(f). \quad (3.2)$$

Це призводить до процедури, яка може знайти мінімум складних неопуклих функцій з відносно невеликою кількістю обрахунків цільової функції, ціною виконання більшої кількості обчисленнь для визначення наступної точки, яку слід спробувати. Коли оцінювання цільової функції є дорогим для виконання - як це відбувається у випадку, коли воно вимагає тренування алгоритму машинного навчання - тоді легко виправдати деякі додаткові обчислення для прийняття кращих рішень (див. рис. 3.1) [8].

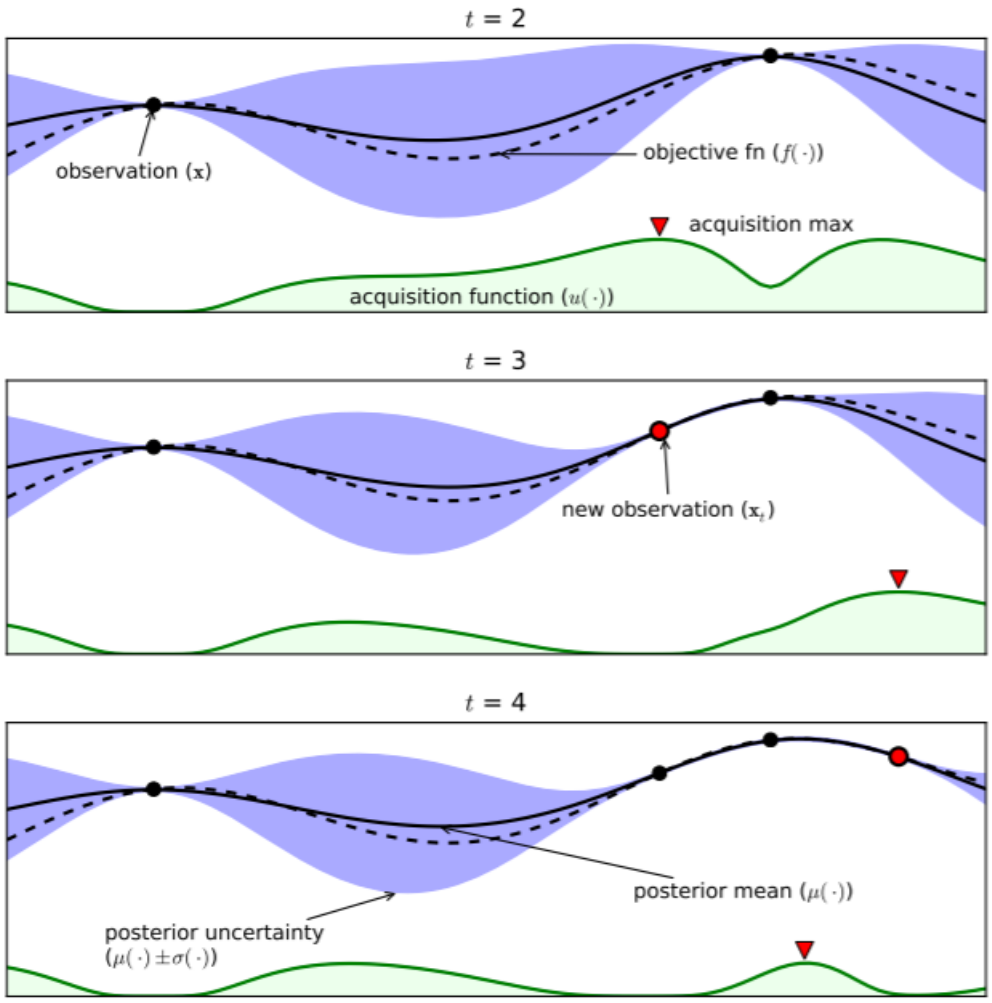

Рисунок 3.1 – Приклад використання байєсівської оптимізації для одномірного випадку

Гауссівські процеси давно визнані хорошим методом моделювання функцій втрат в літературі з оптимізації. Характеристики гауссівських процесів, а також той факт, що гауссівські процеси забезпечують оцінку невизначеності прогнозування цільової функції, роблять цей підхід дуже влучним для знаходження набору гіперпараметрів для подальшого обчислення цільової функції. Час виконання кожної ітерації підходу на гауссівських процесах росте кубічно від кількості попередніх обчислень цільової функції, що треба врахувати для поточної оцінки, та лінійно від кількості змінних що оптимізуються, однак витрати на обчислення цільової функції іноді істотно перевищують навіть цю кубічну вартість [2].

Кубічна обчислювальна складність прогнозування є відомим вузьким місцем передбачення гауссівських процесів, це обмежує їх використання за умови великої кількості обчисленнь цільової функції. Ведуться дослідження над розрідженими гауссовими процесами, які зазвичай базуються на ідеї побудови репрезентативного набору для моделювання даного процесу з загальної вибірки доступних історичних даних.

3.2 Програмне забезпечення для оптимізації гіперпараметрів моделей машинного навчання

На даний момент наявно кілька реалізацій наведених вище методів. Здебільшого це бібліотеки для мови програмування Python, такі як scikit-learn, auto-sklearn, tune, hyperopt, nevergrad. Також іноді використовують промислові системи пристосовані для використання з певними хмарними технологіями, наприклад katib. Ще існують окремі платформи, такі як Amazon SageMaker, Comet.ml, тощо [9].

Sklearn (scikit-learn) – популярна бібліотека, яка надає реалізації деяких моделей машинного навчання. Передбачає два загальних підходи до вибірки наборів параметрів на обчислення цільової функції: для заданих значень GridSearchCV вичерпно розглядає всі комбінації параметрів, тоді як RandomizedSearchCV може вибирати задану кількість кандидатів з простору гіперпараметрів за заданим розподілом. Пошук за сіткою, наданий GridSearchCV, вичерпно генерує кандидатів із сітки значень параметрів, визначених параметром param_grid. При цьому допускається задання кількох сіток гіперпараметрів. В такому разі бібліотека визначить об’єднання множин варіантів гіперпараметрів і так само розгляне усі можливі випадки. GridSearchCV реалізує звичайний API оцінювача, притаманний даній бібліотеці: під час "тренування" його з певним набором даних оцінюються всі можливі комбінації значень гіперпараметрів, а зберігається найкраща комбінація. RandomizedSearchCV реалізує випадковий пошук гіперпараметрів, де кожен гіперпараметр відбирається з розподілу за можливими значеннями параметрів. Це має дві основні переваги над вичерпним пошуком. По-перше, можна задати кількість

ітерацій, яка не буде залежати від кількості параметрів та можливих значень. По-друге, додавання параметрів, які не впливають на продуктивність, не буде знижувати ефективність методу. Визначення вибірки параметрів проводиться за допомогою словника, дуже подібного до визначення параметрів для GridSearchCV. Крім того, кількість вибіркових вибіркових ітерацій (тобто розглянутих варіантів), визначається за допомогою параметра n_iter. Для кожного параметра може бути вказаний розподіл можливих значеннь або список дискретних варіантів вибору (які будуть вибиратися рівномірно). Розподіли певних параметрів можуть бути довільними, також свій розподіл можна імплементувати особисто. До уваги рекомендуються статистичні розподіли з модуля scipy.stats. GridSearchCV та RandomizedSearchCV дозволяють задавати кілька показників (метрик) для оцінки набора гіперпараметрів. Мультиметричне оцінювання може бути вказане як перелік заздалегідь визначених імен об’єктів-оцінювачів або словник, який відображає ім'я оцінювача з функцією оцінювача та/або визначеними за замовчуванням іменами оцінювачів. GridSearchCV і RandomizedSearchCV оцінюють кожен набір гіперпараметрів незалежно. Обчислення можна проводити паралельно, якщо ваша ОС підтримує це, використовуючи аргумент n_jobs=-1 [10].

Auto-sklearn – шар байєсівської оптимізації гіперпараметрів надбудований поверх бібліотеки sklearn. Окрім байєсівського підходу реалізує складання ансамблів з найкращих моделей, отриманих на пройдених кроках оптимізації гіперпараметрів.

Tune – бібліотека Python для виконання оптимізації гіперпараметрів, що є інтегрована та дозволяє масштабувати багато існуючих бібліотек з оптимізації гіперпараметрів. Ця бібліотека підтримує наступні алгоритми та бібліотеки: Grid Search та Random Search, BayesOpt, HyperOpt, SigOpt, Nevergrad, Scikit-Optimize, Ax, BOHB.

Hyperopt – бібліотека Python для послідовної та паралельної оптимізації в специфічних просторах пошуку, що може включати в себе дійсні, дискретні значення та категорії. Ця бібліотека імплементує три алгоритми: випадковий пошук, Tree of Parzen Estimators (TPE), Adaptive TPE [11].

Nevergrad – пакет, що надає наступні рішення:

а) алгоритми оптимізації без градієнтів / похідних, включаючи алгоритми, здатні обробляти шум;

б) інструменти для параметризації будь-якого коду, що робить безболісним оптимізацію ваших параметрів/гіперпараметрів, незалежно від того, чи є вони неперервними, дискретними або сумішшю неперервних та дискретних параметрів;

в) функції, за допомогою яких можна перевірити алгоритми оптимізації;

г) контрольні процедури для порівняння алгоритмів.

Структура пакета nevergrad розбита на наступні підпакети:

- optimization: реалізація алгоритмів оптимізації;
- parametrization: визначення, які саме параметри потрібно оптимізувати;
- functions: реалізація як простих, так і складних функцій експериментів;
- benchmark: для запуску експериментів, для порівняння алгоритмів на функціях експериментів;
- common: набір загальних інструментів, які використовуються у всьому пакеті;

Nevergrad дозволяє проводити оптимізацію паралельно, з довільною кількістю виконавців (проте не всі алгоритми оптимізації підтримують таку паралелізацію)

Katib – це система, розроблена для використання з Kubernetes для оптимізації гіперпараметрів та пошуку архітектури нейронних мереж. Візуалізація результатів у даній системі здебільшо пристосована саме до цієї предметної області, як показано на рисунку 3.2. Katib підтримує ряд фреймворків машинного навчання, включаючи TensorFlow, Apache MXNet, PyTorch, XGBoost та інші. Для задачі оптимізації гіперпараметрів використовуються наступні методи: випадковий пошук, пошук по сітці, Tree of Parzen Estimators (TPE), Hyperband, Bayesian Optimization.

Крім налаштування гіперпараметрів, Katib пропонує функцію пошуку нейронної архітектури (англ. NAS, Neural Architecture Search). NAS можна використовувати для розробки моделей машинного навчання на основі штучних нейронних мереж з метою досягнення максимальної точності прогнозування та продуктивності моделі [12].

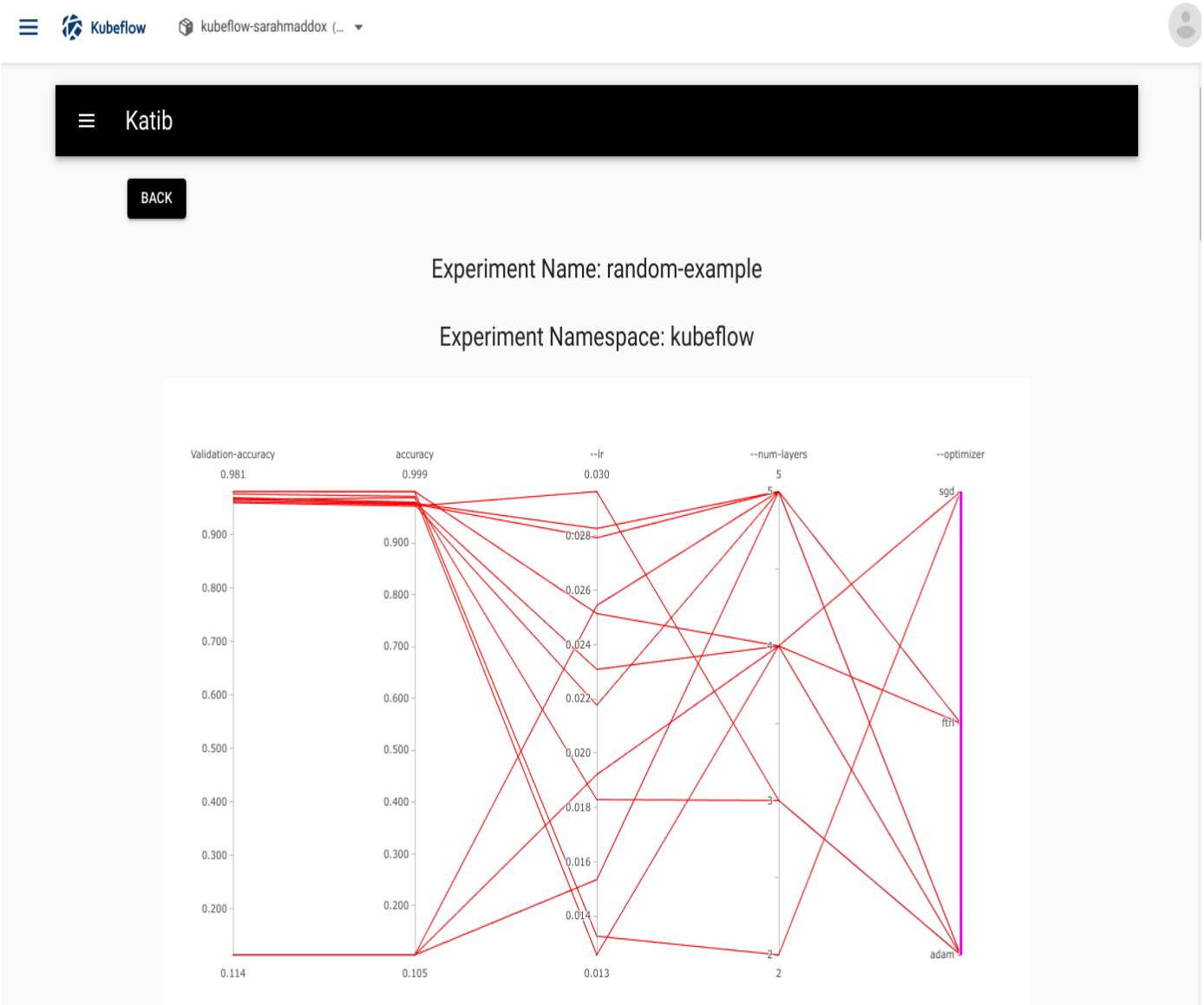

Рисунок 3.2 – Графічний інтерфейс katib, демонструється точність моделі манинного навчання за різних наборів гіперпараметрів

Такі продукти, як katib, зазвичай є end-to-end системами та надають можливість складних налаштуваннь, добре масштабуються, керують наданими ресурсами. Хоча достиь часто вони не дозволяють динамічно змінювати вже запущений процес оптимізації і прив’язані до певних хмарних платформ чи технологій. Створення експериментів у подібних системах часто має достатньо зручний користувацький інтерфейс, як показано на рисунку 3.3. Ці системи пристосовано для використання у масштабах кластерів для обчисленнь.

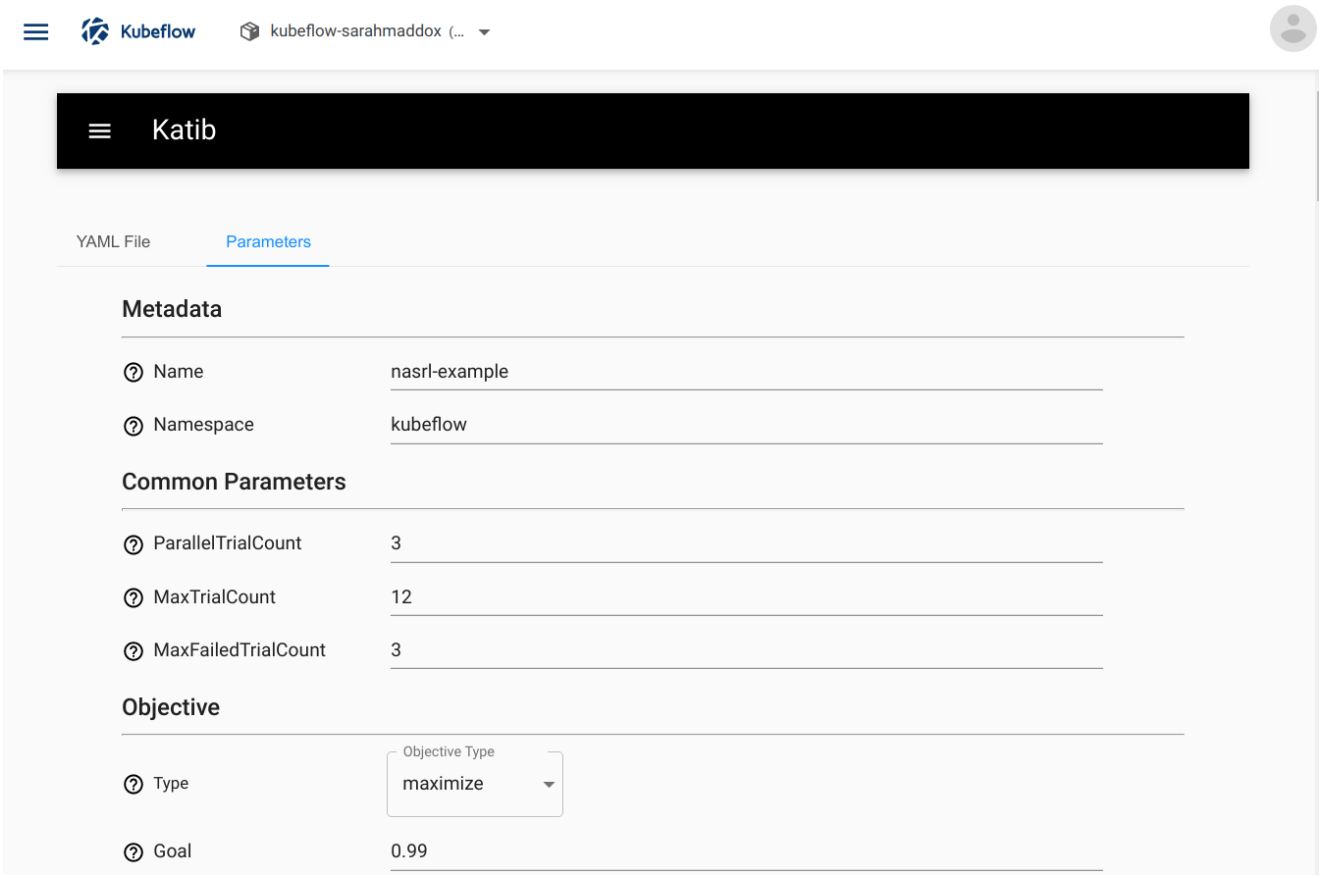

Рисунок 3.3 – Приклад інтерфейсу для створення експерименту у системі katib

В програмному комплексі katib реалізовано зручний моніторинг створених експериментів, як активних, так і вже завершених (див. рис. 3.4).

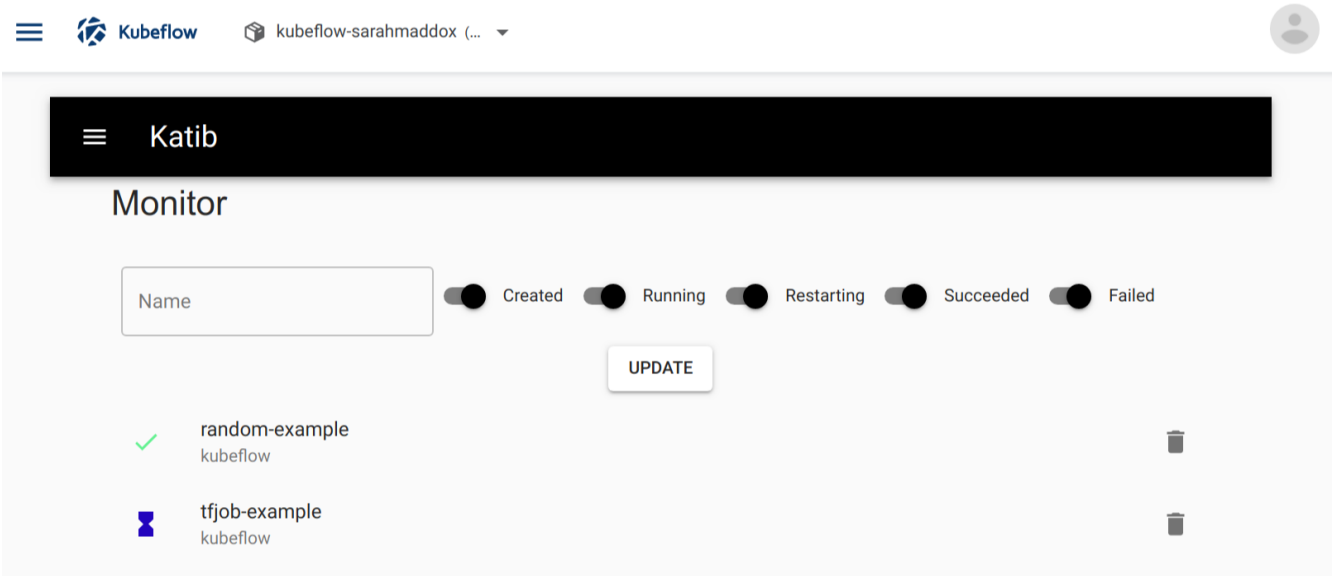

Рисунок 3.4 – Моніторинг експериментів у системі katib

Іноді використовуються платформи, що націлені на задачі машинного навчання. Comet.ml є яскравим прикладом такої платформи. Вона дозволяє користувачеві оперувати багатьма проектами з кількома експериментами одночасно, а також зберігає інформацію про старі експерименти (див. рис. 3.5).

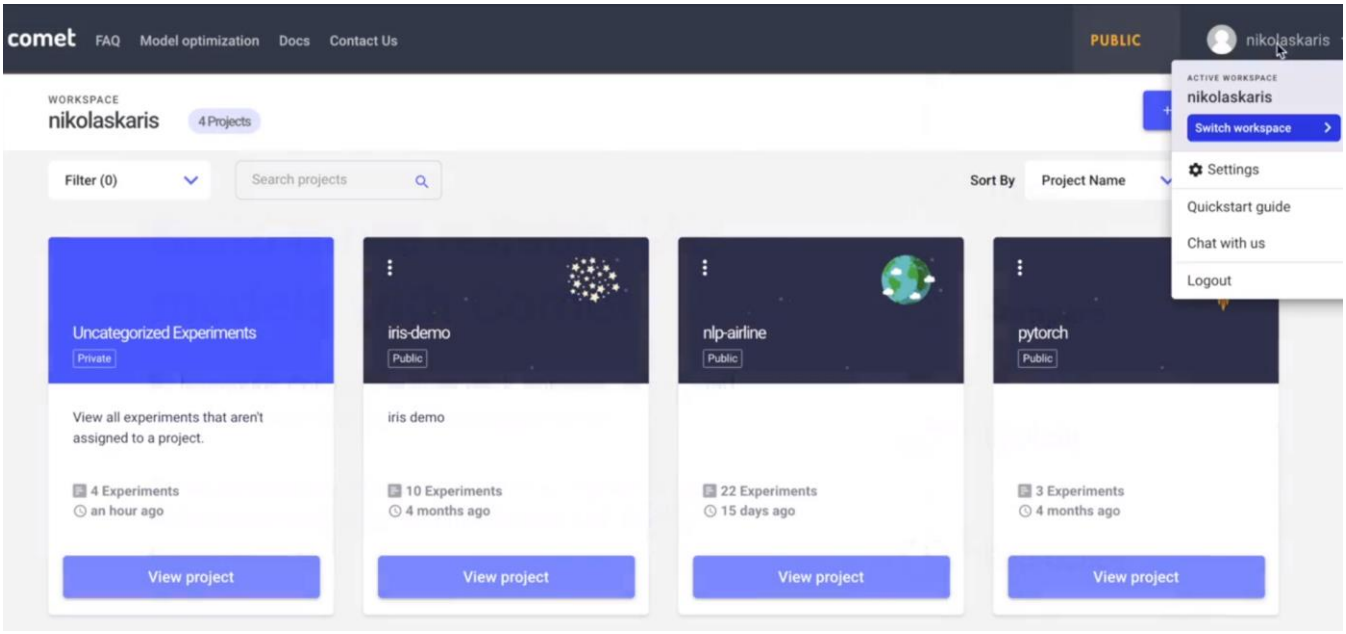

Рисунок 3.5 – Керування проектами на платформі Comet.ml

Платформа надає графічний інтерфейс з даними експериментів (див. рис. 3.6).

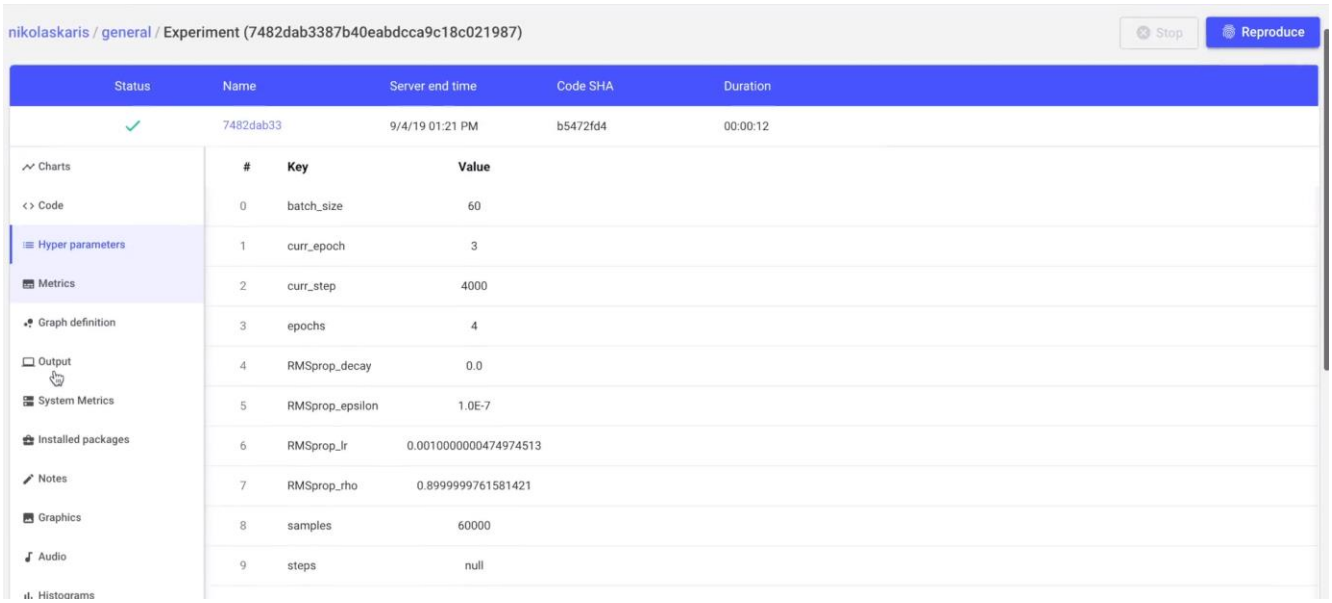

Рисунок 3.6 – Відомості про результати експерименту у Comet.ml

Платформа також позволяє візуалізувати значення цільової функції, та порівнювати хід різних варіацій експериментів (див. рис. 3.7)

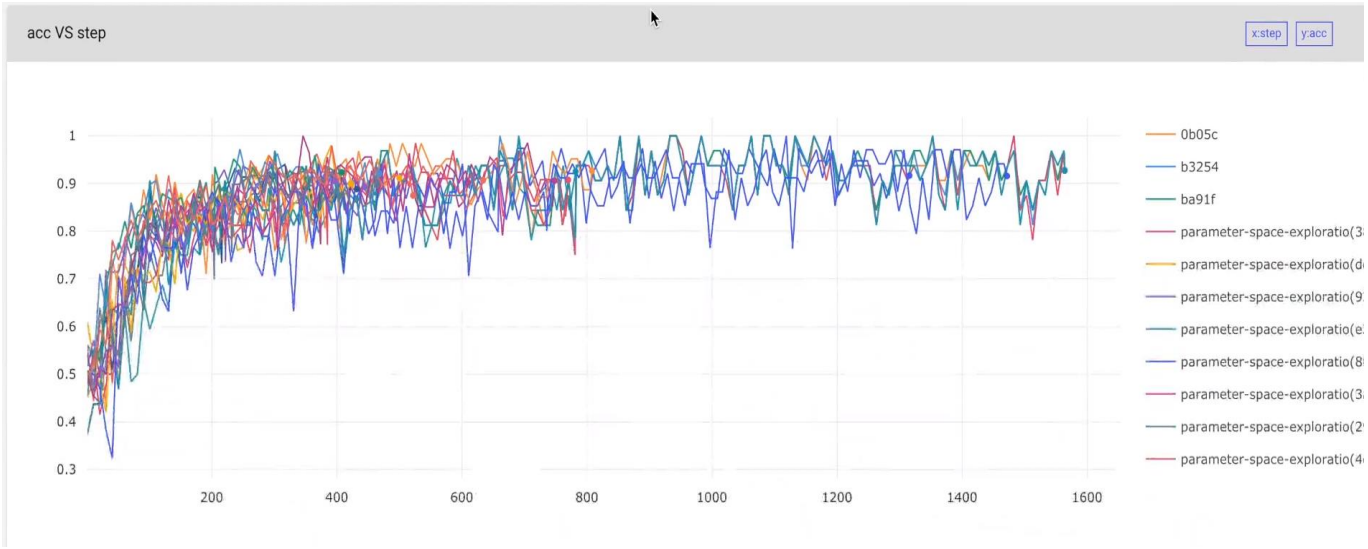

Рисунок 3.7 – Візуалізація точності моделі зі зміною кроку ітерації у Comet.ml

Comet.ml також дозволяє оперувати кількома метріками та відображати результати таких експериментів на кількох графіках поруч (див. рис. 3.8).

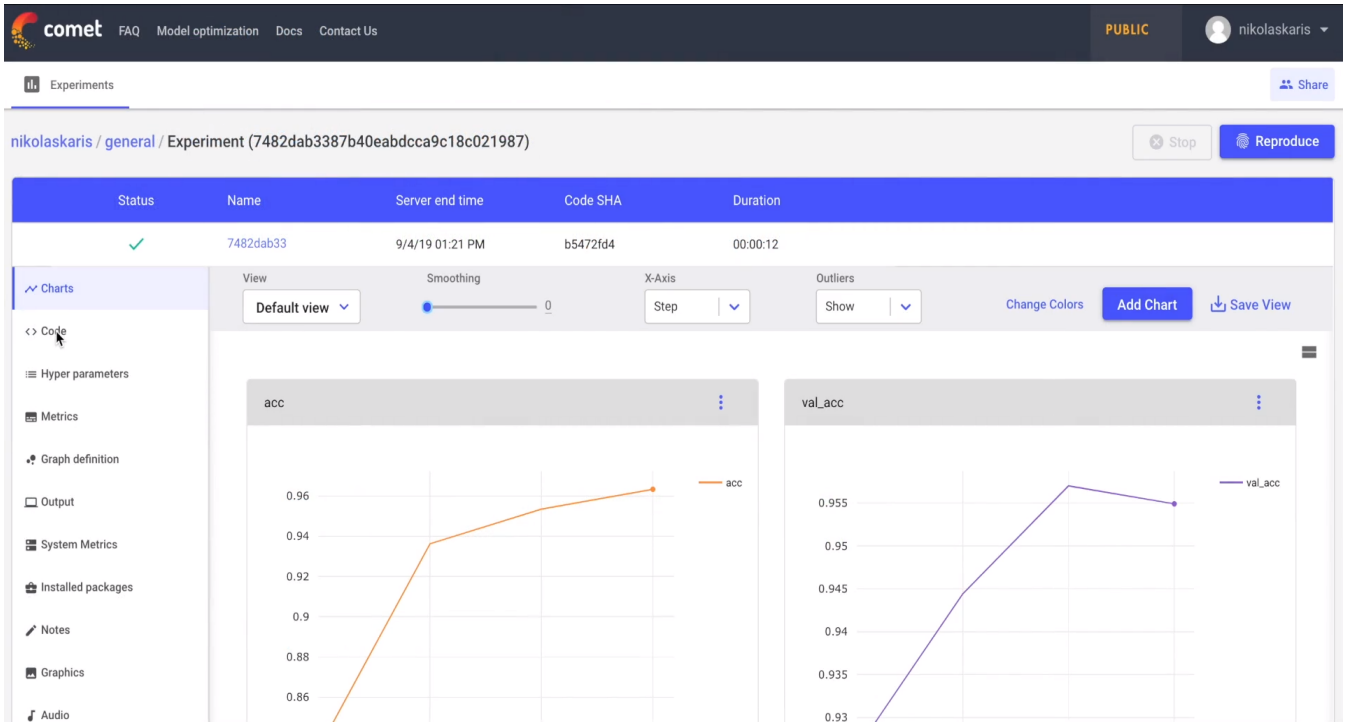

Рисунок 3.8 – Використання кількох метрік на платформі Comet.ml

Платформа Comet.ml також надає можливість зберігати та відображати користувачеві дані, на яких тренується та потім валідується модель машинного навчання. Зокрема, при роботі з графічними зображеннями можна переглянути результати натренованої моделі на певних контрольних прикладах, що користувач загружає у окрему форму веб-інтерфейсу (див. рис. 3.9).

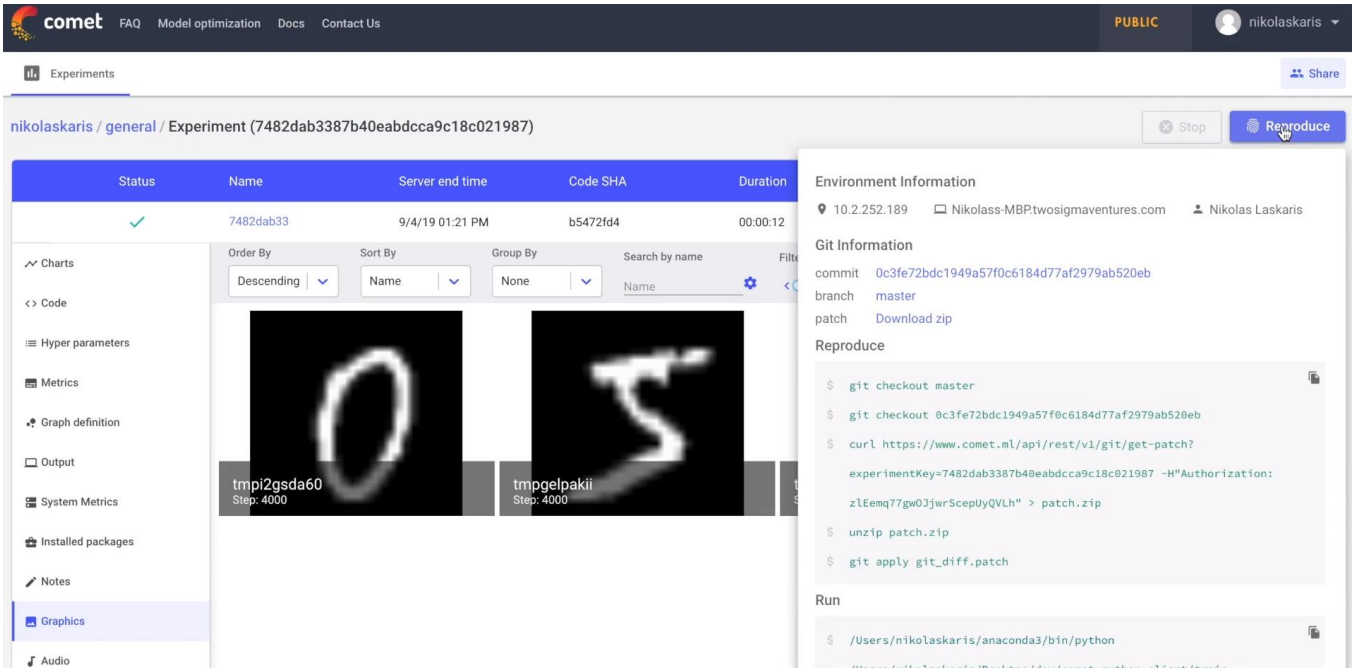

Рисунок 3.9 – Зберігання контрольних прикладів користувацького набору даних

На платформах які надають можливості по мінімізації похибок моделей машинного навчання за допомогою вибору оптимальних гіперпараметрів зазвичай дуже зручно реалізовано керування проектами та окремими експериментами в рамках проекту (див. рис. 3.10). Такі платформи надають графічний інтерфейс для більшості функцій, що доступні у вищеописаних бібліотеках чи системах на базі хмарних середовищ . Додатковою перевагою подібних платформ є відсутність необхідності виділення ресурсів на проведення експериментів. Частіше за все платформи для роботи з моделями машинного навчання виступають у ролі хмарного сервісу та беруть на себе питання виділення та масштабування обчислювальних ресурсів. Це дозволяє користувачам не турбуватися про тимчасові збільшення обчислювальних

ресурсів, проте інтенсивне використання подібних сервісів може бути значно дорожче за покупку звичайних хмарних ресурсів.

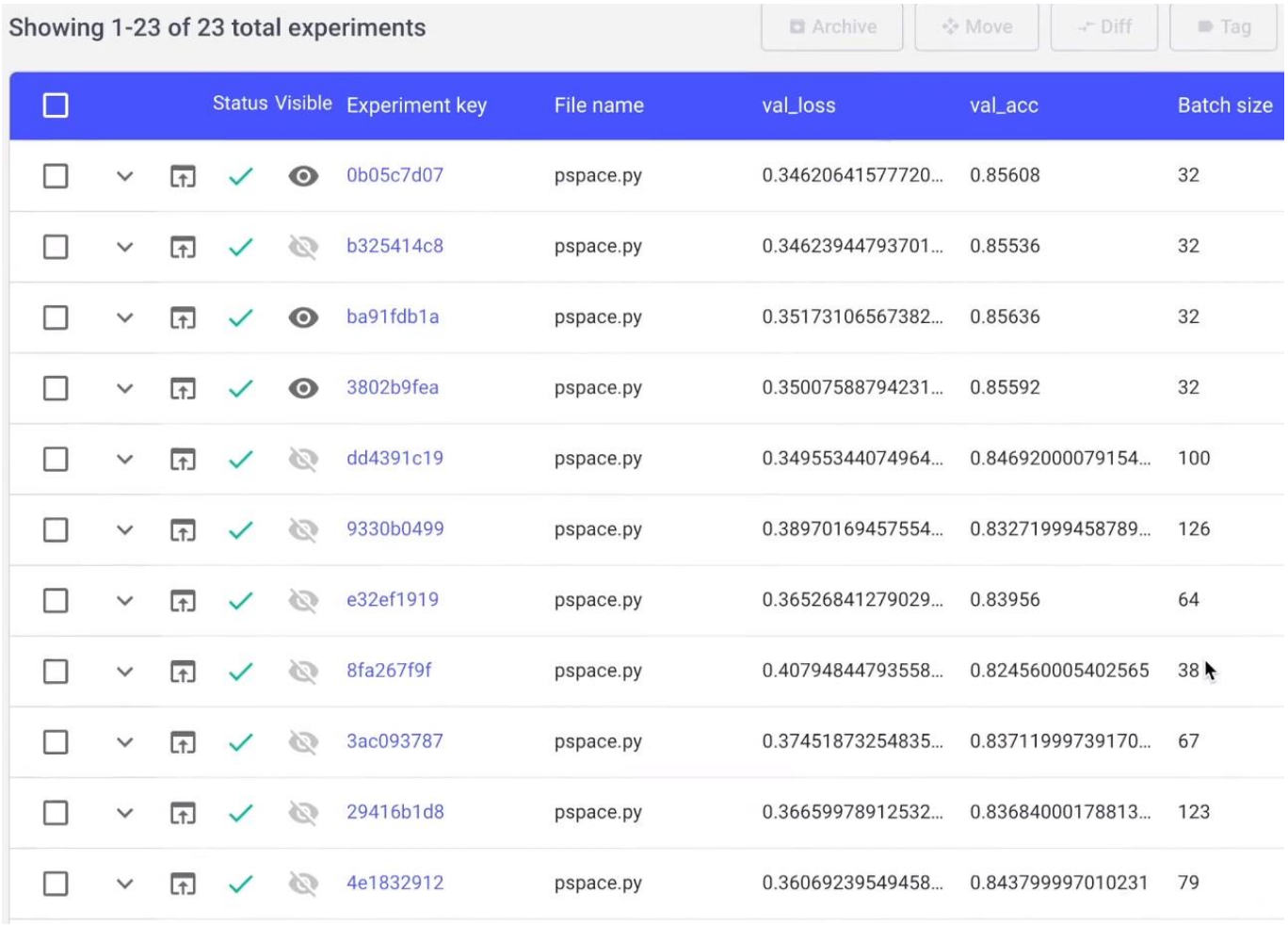

Рисунок 3.10 – Керування експериментами у рамках проекту Comet.ml

Подібні платформи вимагають виконання розрахунків і збереження користувацьких даних на їх стороні, що може бути не лише перевагою, але й стає значною перешкодою для підприємств з підвищеними вимогами до безпеки даних.

Таким чином, актуальною задачею є розробка сервісу для оптимізації гіперпараметрів, що має можливість розгортання на стороні користувача, є незалежним від мови програмування чи хмарних технологій користувача і надає прикладний інтерфейс для взаємодії з клієнтськими програмними засобами.

4 ВИБІР ЗАСОБІВ РОЗРОБКИ

Мовою програмування для реалізації даного проекту обрано Python. Ця мова має серед переваг сильну динамічну типізацію, що полегшує роботу у просторі довільних гіперпараметрів без необхідності створювати складну структуру абстракцій для роботи з параметрами різних типів у просторі довільної розмірності. При цьому факт що типізація у мові програмування Python є сильною збереже від багатьох потенційних помилок пов’язаних з приведенням типів [13]. Додатковою перевагою Python є широкий набір стандартних бібліотек. Багато завдань програмування з високим рівнем використання вже імплементовано у стандартній бібліотеці, що значно скорочує довжину коду для написання власних продуктів [14].

Серед фреймворків для полегшення написання прикладного програмного інтерфейсу було обрано Flask, через популярність і простоту створення додатків [15].

Для полегшення розробки сервісу, а також спрощення подальшої підтримки та вдосконалення цього рішення рекомендовано використовувати систему контролю версій для відстеження змін у вихідному коді продукту [16]. Також така система дозволить безболісно долучати інших розробників до процесу вдосконалення продукту. Система контролю версій дозволяє користувачам відслідковувати зміни в проектах з розробки програмного забезпечення та дає їм змогу співпрацювати над цими проектами. Використовуючи таку систему, розробники можуть спільно працювати над кодом та розділяти свої завдання через створення різних гілок розробки. У системі управління версіями може бути кілька гілок відповідно до кількості співпрацівників. Гілки зберігають індивідуальність, оскільки зміни коду залишаються у визначеній гілці. Розробники можуть комбінувати внесенні до коду зміни за потреби. Далі вони можуть переглянути історію змін, повернутися до попередніх версій та використовувати/керувати кодом бажаним чином. Основними перевагами використання системи контролю версіями є впорядкування процесу розробки, управління кодом для декількох проектів та збереження історії всіх змін у коді. Програмне забезпечення для управління версіями зберігає всі зміни в сховищі. Отже, якщо розробники допустили помилку, вони можуть її скасувати. У той же час

вони можуть порівнювати новий код з попередньою версією, щоб вирішити нові проблеми. Це може значно зменшити наслідки людської помилки чи ненавмисного припинення працездатності сервісу. Крім того, система контролю версій може бути інтегрованою з певними інструментами розробки програмного забезпечення, такими як PaaS-постачальниками (платформи для розгортки продукту), IDE (інтегрованими середовищами розробки) та інструментами автоматизації збірки проекту [17].

Наразі існує багато альтернативних систем контролю версій, таких як Git, Mercurial, CVS, Subversion та інші. Через наявність досвіду роботи та широку популярність, для використання у даному проекті було обрано систему контролю версій Git. Також, використання Git як системи контролю версій дозволяє зручно відкрити та поширювати доступ до вихідного коду сервісу за допомогою платформи GitHub, де для даного проекту створено свій репозиторій.

Задля подальшого розповсюдження сервісу, його образ було розміщено на платформі Docker Hub. Docker Hub – це хмарне сховище, в якому користувачі та партнери Docker створюють, тестують, зберігають та розповсюджують образи контейнерів. Через Docker Hub користувач може отримувати доступ до загальнодоступних сховищ образів з відкритим кодом, а також використовувати простір для створення власних приватних сховищ, автоматизації збирання проектів та інше [18].

Тестування сервісу вимогало інтерактивного середовища розробки, у якому було би зручно власноруч виконувати певні ділянки коду, відправляти сервісу запити, генерувати вхідні дані та обробляти повідомлення що прийшли у відповідь на запити. У якості такого середовища було обрано Jupyter Notebook (надалі “ноутбук”). Jupyter Notebook розширює консольний підхід до інтерактивних обчислень у якісно новому напрямку впроваджуючи веб-додаток, придатний для фіксації всього процесу обчислень: розробки, документування та виконання коду, а також передачі результатів. Jupyter Notebook поєднує два компоненти:

- веб-додаток: інструмент на основі браузера для інтерактивного складання документів, що поєднують пояснювальний текст, математику, обчислення та їх багатий медіа-вихід;

- ноутбук-документи (файли, що зазвичай мають розширення .ipynb): представлення всього вмісту, видимого у веб-додатку, включаючи входи та виходи обчислень, пояснювальний текст, математику, зображення та мультимедійні зображення об’єктів.

Основні можливості, що надає веб-додаток системи Jupyter Notebook користувачеві:

- редагування коду безпосередньо у браузері з автоматичним підсвічуванням синтаксису, та доповненням коду по натисканню Tab, або інспекцією доступних методів, атрибутів, тощо по натисканню тієї ж клавіші;
- можливість виконання коду з браузера, при цьому результати обчислень додаються до коду, який їх згенерував та доступні для подальшого використання;
- відображення результату обчислень за допомогою мультимедійних представлень, таких як HTML, LaTeX, PNG, SVG тощо. Наприклад, графіки, що генеруються за допомогою бібліотеки matplotlib, можуть бути включені в рядок ноутбуку та подалі можуть використовуватися у наукових публікаціях, рефератах, статтях, тощо;
- внутрішнє браузерне редагування тексту насиченого спеціальними символами та стилями за допомогою мови розмітки Markdown. Таким чином можна надавати коментарі до коду у одному файлі з власне вихідним кодом, не обмежуючись простим текстом;
- можливість легко включати математичні позначення всередині комірок розмітки за допомогою LaTeX. Такі включення рендеряться за допомогою MathJax.

Ноутбук-документи містять власне вихідний код доступний для виконання, дані що подаються на вхід та отримуються з виходу інтерактивного сеансу виконання коду, а також додатковий текст, який супроводжує код, але не призначений для виконання. Таким чином, ноутбук-файли можуть слугувати повноцінним записом обчислювального сеансу, поєднуючи виконуваний код із пояснювальним текстом, математичними даними та медіа-відображеннями отриманих об'єктів. Ці документи

по внутрішній структурі є файлами JSON і зберігаються з розширенням .ipynb. Оскільки JSON - це звичайний текстовий формат, для таких файлів можна легко керувати версіями файлу при сумісній роботі з колегами. Системи контролю версій будуть відслідковувати зміни так само, як і у звичайних текстових файлах вихідного коду більшості мов програмування.

Ноутбуки можуть бути експортовані до цілого діапазону статичних форматів, включаючи HTML (наприклад, для публікацій у власному блозі), reStructuredText, LaTeX, PDF та до деяких популярних форматів слайд-шоу за допомогою команди nbconvert.

Крім того, будь-який ноутбук-документ, доступний за загальнодоступною URL-адресою, може бути відкритий для просмотру через засіб перегляду ноутбуків Jupyter Notebook Viewer (nbviewer). Цей сервіс завантажує документ-ноутбук за URL-адресою та надає його користувачеві як статичну веб-сторінку. Таким чином, результати можуть бути опубліковані для загального доступу, для доступу колег або як публічна публікація в блозі, причому іншим користувачам не потрібно самостійно встановлювати Jupyter Notebook як додаток на власну обчислювальну машину. Фактично, nbviewer просто використовує утіліту nbconvert як веб-сервіс, так само як користувачі можуть робити власні перетворення ноутбук-документів у статичний формат за допомогою nbconvert, не покладаючись на nbviewer.

Оскільки Jupyter Notebook зазвичай використовується у веб-браузері, деякі користувачі, зрозуміло, стурбовані його використанням із залученням конфіденційних даних. Однак якщо дотримуватись стандартних інструкцій із встановлення, Jupyter Notebook насправді працює на власному комп’ютері користувача і не передає дані на сторонні сервери, якщо такі дії не запрограмовано у виконуваному коді користувача. Якщо URL-адреса в адресному рядку браузера, за якою відбувається доступ до запущеного ноутбуку починається з http://localhost: або http://127.0.0.1: то це свідчить, що комп'ютер користувача виступає в якості сервера. Jupyter Notebook не надсилає користувацькі дані у будь-які інші місця і, оскільки цей проект знаходиться у відкритому доступі, інші люди можуть перевірити вихідний код

сервісу задля того щоб упевнитися у відсутності команд на передачу даних третім особам.

Також Jupyter Notebook надає можливість використовувати ноутбуки дистанційно: наприклад, компанія або університет може запустити сервер для своїх користувачів. У цьому разі задля того, щоб працювати з конфіденційними даними треба забезпечувати безпеку взаємодії з системою Jupyter Notebook через мережеве з'єднання самостійно.

Команда розробників Jupyter Notebook має на меті забезпечення засобів безпеки, щоб інші сторінки у вашому веб-браузері чи інші користувачі на тому ж комп'ютері не мали доступу до вашого сервера ноутбуків. Докладніше про це описано у документації на офіційному сайті Jupyter Notebook.

Недоліками системи Jupyter Notebook можна зазначити дещо лімітовані інструменти забезпечення підказок при написанні коду, особливо порівняно з розвинутими популярними інтегрованими середовищами розробки. Наприклад, для мови програмування Python деякі середовища розробки, наприклад PyCharm, дозволяють використовувати статичну перевірку типів змінних, аргументів функцій, користувацьких класів, тощо. У той час підказки пов'язані з типізацією у Jupyter ноутбуках відсутні за замовчуванням.

Проте задля написання тестових та демонстраційних скриптів мовою програмування Python, система Jupyter Notebook є дуже зручною у використанні. Зручність задля досягнення цієї мети обумовлена можливістю поділу коду на комірки, збереження результатів виконання окремих комірок видимим для користувачів, що надалі переглядають ноутбук, наскрізного коментування та документації коду з ноутбуку за допомогою як тексту так і медіа-контенту. Всі ці можливості переважають зазначені недоліки коли мова йдеться про створення демонстраційного скрипту, тому у даній роботі саме Jupyter Notebook використовується задля демонстрації можливостей взаємодії користувача зі створеним сервісом.

5 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Виходячи з обгрунтування, наведеного у розділі 4, програмний продукт необхідно розробити як сервіс, на мові програмування Python з використанням фреймворку Flask. Сервіс має надавати прикладний програмний інтерфейс (Application Programming Interface, API), що дозволяє проводити оптимізацію гіперпараметрів машинного навчання, посилаючи HTTP запити з будь-якого користувацького програмного продукту чи середовища до сервісу. Також сервіс має надавати графічний web-інтерфейс для наочного відображення стану оптимізації, переліку створених експериментів по оптимізації гіперпараметрів, перегляду історії кроків оптимізації.

Для зручності опису інтерфейсу, введемо поняття експерименту, як процесу оптимізації гіперпараметрів з одного заданого простору для однієї моделі машинного навчання, при якому сервіс має використовувати обчислені значення цільової функції (у даному випадку метріки, що показує точність отриманої моделі) у наступних ітераціях оптимізації.

Для виконання оптимізації гіперпараметрів у сервісі має бути доступно декілька оптимізаторів. Оптимізатором є клас, що реалізує певний алгоритм мінімізації цільової функції. В даній програмній реалізації будуть доступні оптимізатори що реалізують Баєсівську оптимізацію, генетичні алгоритми та випадковий пошук.

5.1 Алгоритм взаємодії з сервісом

Життєвий цикл сервісу може підтримувати одна людина, у найпростішій конфігурації за замовчуванням. Проте оперування сервісом може бути полегшено у разі розподілу операцій між фахівцем що знається на науці про дані та інженером даних (див. рис. 5.1). Зокрема, такі функції програмного рішення, як розгортка сервісу, збірка та налаштування власного docker-образу сервісу, конфігурація ресурсів та мережевих налаштувань для успішної взаємодії з сервісом може

проводитися інженером даних. Тоді на фахівця, що працює з прототипізацією моделей машинного навчання залишається власне створення та редагування експериментів (деяка частина цих процедур може бути оптимізована інженером даних), а також контроль якості отриманих результатів, валідація та верифікація отриманих даних. За потреби фахівець науки про дані може реалізувати власні алгоритми оптимізації для використання під час роботи з сервісом.

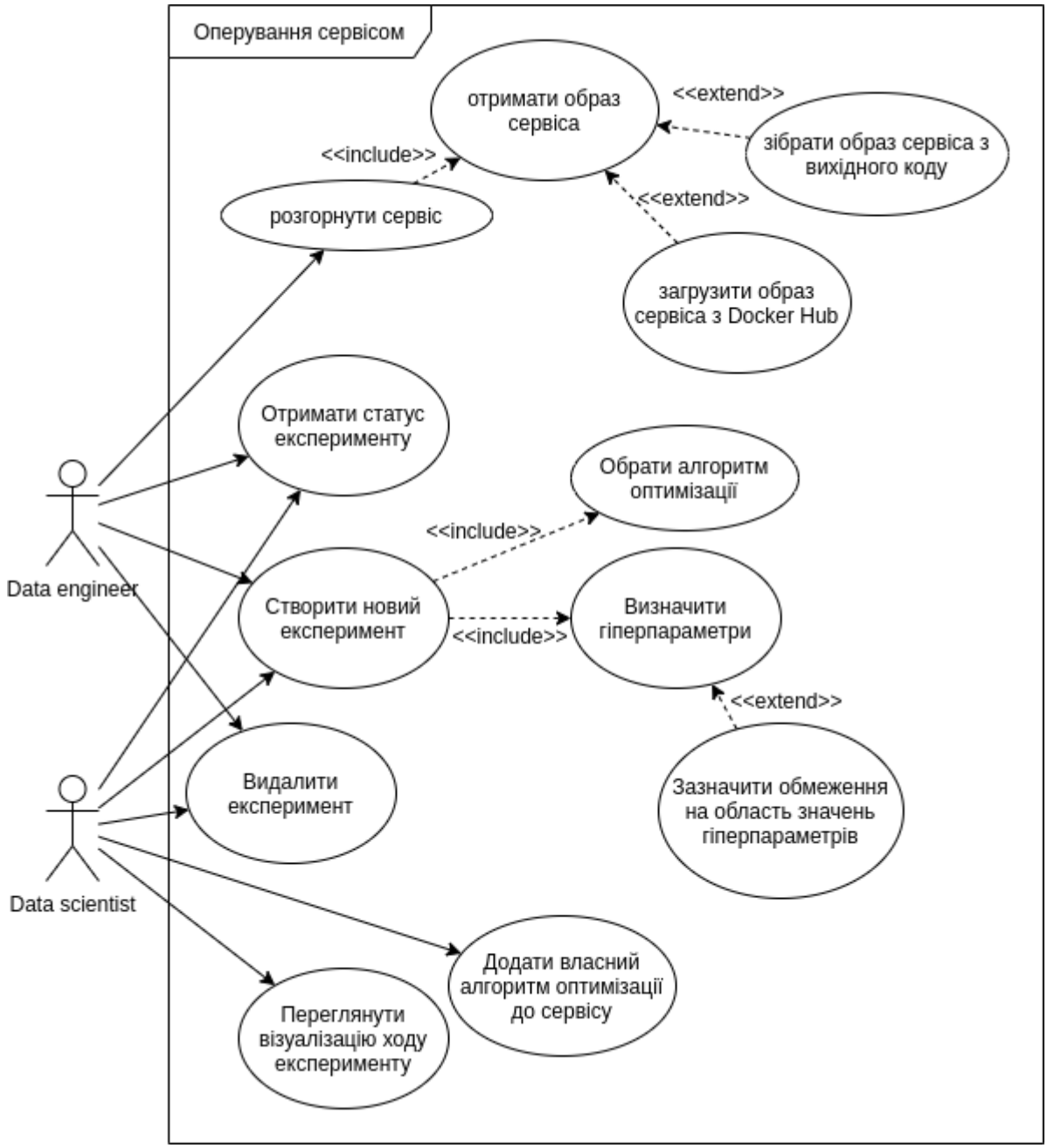

Рисунок 5.1 Use case діаграма оперування сервісом

Успішна взаємодія зі створеним сервісом передбачає дотримання певної послідовності дій (див. рис. 5.2). Так як сервіс створений задля оптимізації гіперпараметрів моделей машинного навчання, користувач має обрати модель машинного навчання та цільову метріку задля оцінки обраної моделі перед початком роботи з сервісом. Далі слід обрати набір гіперпараметрів що оптимізуються.

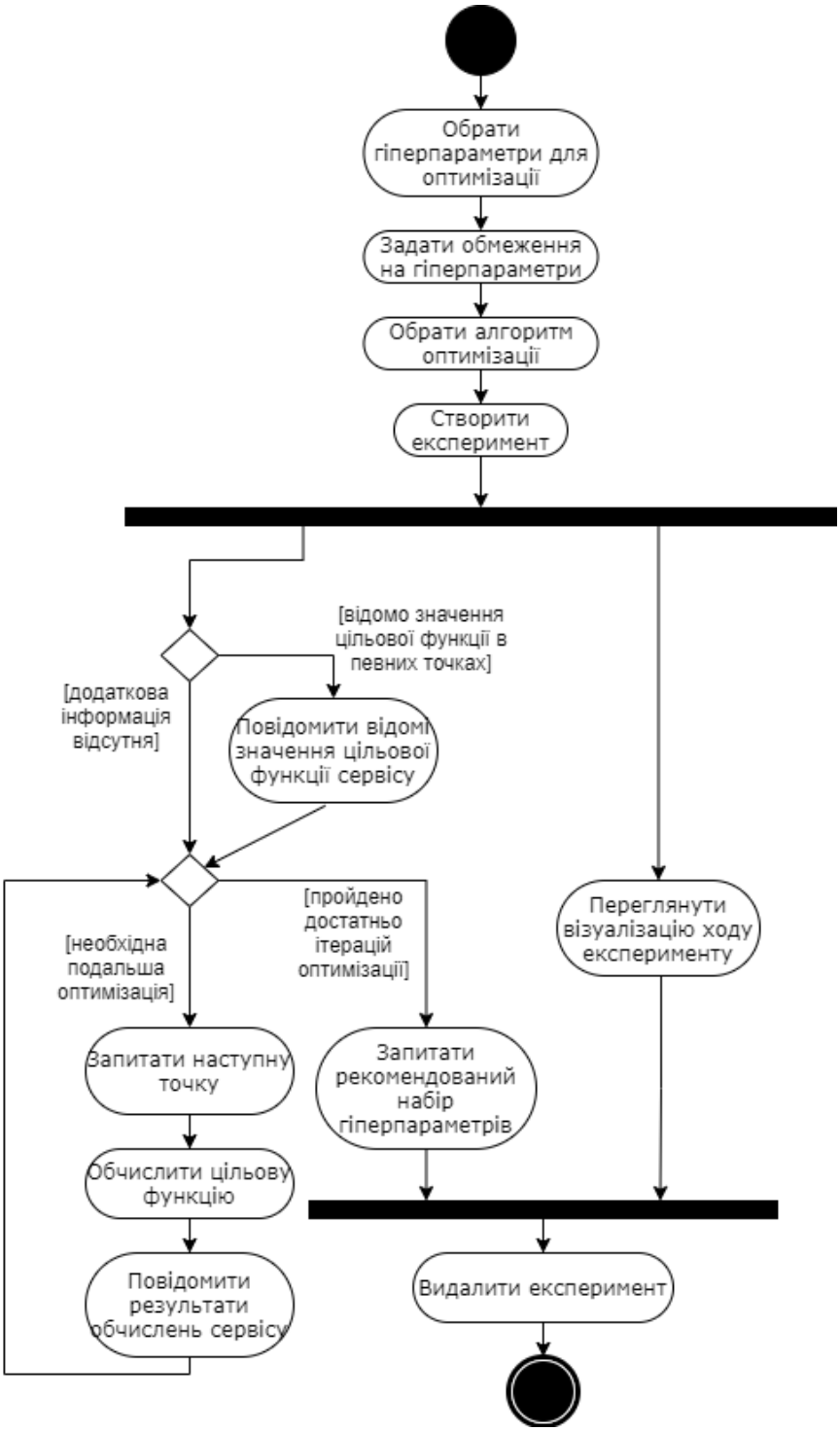

Рисунок 5.2 — Діаграма діяльності життєвого циклу експерименту

Для кожного з гіперпараметрів треба визначити тип та обмеження. У сервісі доступні такі типи:

- чисельний (scalar) з можливістю встановлення верхньої і нижньої границь на значення та можливістю використовувати лише цілі значення;
- чисельний з логарифмічним розподілом (log) з обов'язковим зазначенням границь;
- тип вибору (choice) що дозволяє обирати одне значення з переліку допустимих варіантів (перелік варіантів вважається невпорядкованим).

Наступним кроком треба обрати алгоритм оптимізації. Рекомендовано обирати Байєсівську оптимізацію (назва оптимізатору в API - BO), коли необхідно досягти швидкої збіжності алгоритму, проте слід зазначити що цьому алгоритму притаманна доволі сильна скупученість точок що перевіряються, що може призвести до великої кількості обчислень поблизу локальних мінімумів цільової функції. Для більш гнучкої оптимізації можна обрати генетичний алгоритм OnePlusOne. Задля рівномірного огляду простору гіперпараметрів можна обрати випадковий пошук (RandomSearch). Далі слід створити експеримент, передавши до сервісу усі означені параметри.

Сервіс надає змогу повідомити відомі значення цільової функції при певних значеннях гіперпараметрів для подальшого використання при оптимізації. Для проведення оптимізації за обраним алгоритмом необхідно циклічно відправляти запит на сервіс задля отримання пропонованого набору гіперпараметрів, обчислювати значення цільової функції при даних значеннях гіперпараметрів та відправляти повідомлення з зазначенням набору гіперпараметрів та відповідного значення цільової функції сервісу. Слід зазначити, що обчислення цільової функції можна проводити паралельно, тобто можна спочатку відправити кілька запитів на рекомендовані точки у просторі гіперпараметрів, а потім відправляти повідомлення зі значеннями цільової функції. За процесом оптимізації можна слідкувати через web-інтерфейс.

Користувач вирішує, коли треба зупиняти процес оптимізації гіперпараметрів на свій розсуд. Це можна робити спираючись на витрачений час та обчислювальні

ресурси, на досягнення певних значень цільової функцій, на характер збіжності значень цільової функцій, тощо. Оптимальний набір гіперпараметрів, знайдений в рамках певного експерименту можна отримати окремим запитом, разом з кількістю досліджених точок. Такий запит можна відправляти у будь-який час існування експерименту. Після завершення роботи з певним експериментом данні про нього можна видалити з сервісу відправивши відповідний запит по API.

5.2 Прикладний програмний інтерфейс (API)

За формат даних для обміну через API було обрано JSON (JavaScript Object Notation). Перевагами цього формату передачі даних є невеликий надлишковий розмір повідомленнь у порівнянні з обсягом власне переданої корисної інформації та можливість передавати дані досить складної структури. Наведені переваги забезпечили популярність формату JSON, а це значить що розробникам буде зручно користуватися створеним сервісом та в них вже з великою вирогідністю буде досвід роботи з даним форматом. Також в інтернеті існує доволі багато інформації про роботу з повідомленнями переданими у цьому форматі, а у багатьох мовах програмування утіліти для роботи з цим форматом передачі данних включені до стандартної бібліотеки.

API сервісу дозволяє створювати декілька експериментів, що може бути корисно якщо користувач хоче оптимізувати гіперпараметри для кількох різних моделей машинного навчання одночасно і незалежно.

Прикладний програмний інтерфейс продукту приймає запити за кількома адресами:

- /status;
- /experiment;
- /experiment/<experiment_id>;
- /experiment/<experiment_id>/ask;
- /experiment/<experiment_id>/tell;

За адресою /status сервіс приймає GET запити без параметрів та відправляє відповідь JSON повідомлення { "status": "OK" }. Рекомендовано для використання при перевірці, що сервіс запущено і він працює.

За адресою /experiment сервіс приймає GET і POST запити. У разі POST запиту потрібно передавати параметри запиту певного формату. На рисунку 5.3 наведено формат даних для запиту на створення сервісу.

```
POST /experiment
{
  "params": {
    <param_name>: {
      "type": "scalar",
      "parameters": {
        "lower": <float_number>,
        "upper": <float_number>
      }
    },
    <param_name>: {
      "type": "choice",
      "parameters": {
        "choices": [<string>, ...]
      }
    },
    ...
  },
  "optimizer": <optimizer_title>
}
```

Рисунок 5.3 – Формат тіла запиту на створення експерименту

Загалом, запит містить вибір оптимізатора та оголошення гіперпараметрів з обмеженнями на область їх значень. Для кожного гіперпараметру <param_name> – це довільна строка що буде використована як ім’я гіперпараметра. У якості типу (type) гіперпараметра можна використовувати такі варіанти: "scalar", "log", "choice".

Тип scalar використовується для гіперпараметрів числового типу, тип log – для параметрів числового типу, розподіл яких у процесі вибору значень для перевірки слід вважати логарифмічним; тип choice – для гіперпараметрів які передбачають вибір значення з певного дискретного набору.

Оголошення кожного з гіперпараметрів має містити обмеження на область значень у полі "parameters". Для гіперпараметрів числових типів ("scalar", "log") обмеження мають включати нижню і верхню границі області значень, та бути представленими як числові значення параметрів "lower" і "upper" відповідно. При цьому для гіперпараметрів типу "log" значення нижньої та верхньої границь мають бути строго більше за нуль. Для типу choice множина можливих значень має бути представлена як список строк у параметрі "choices".

Значення <optimizer_title> має містити строку з назвою оптимізатору. Підтримуються варіанти "BO" (для байєсівської оптимізації), "OnePlusOne" (генетичний алгоритм), "RandomSearch" (випадковий пошук), "GridSearch" (пошук за сіткою з кроком що зменьшується) або назви користувацьких оптимізаторів, якщо вони імплементовані. Запит повертає строку, що є ідентифікатором створеного експерименту.

У разі GET запиту сервіс поверне список ідентифікаторів активних експериментів у наступному форматі: [{ "experiment_id": <experiment_id> }, …], де <experiment_id> – це строка з ідентифікатором експерименту. Ідентифікатор генерується при створенні експерименту за стандартом uuid-4. Експеримент вважається активним з часу як його було створено сервісом за допомогою POST запиту за цією адресою і до моменту видалення експерименту (див. далі).

За адресою /experiment/<experiment_id> сервіс приймає GET та DELETE запити. У разі DELETE запиту експеримент з ідентифікатором <experiment_id> видаляється з сервісу, тобто перестає бути активним, а всі дані щодо нього втрачаються. У відповідь на успішно виконаний DELETE запит повертається повідомлення формату { "status": "OK" }. У разі GET запиту сервіс надає відповідь у форматі, як показано на рисунку 5.4.

```
{
  "n evaluated points": <int_number>,
  "recommended point": {
    <param_name>: <float_number>,
    ...
  }
}
```

Рисунок 5.4 – Формат відповіді на запит статусу експерименту

У даному форматі відповіді <int_number> це ціле число, <float_number> – число з плаваючою точкою, а <param_name> – строка з іменем гіперпараметру. Видно, що статус включає кількість точок, в яких обчислено цільову функцію, а також значення гіперпараметрів у точці, що на поточний час вважається оптимальною.

За адресою /experiment/<experiment_id>/ask сервіс приймає GET запити. У відповідь на такий запит буде повернуто повідомлення формату { <param_name>: <value>, ... }, що містить перелік імен гіперпараметрів (<param_name>) та відповідних їм значень (числових, у разі гіперпараметрів числового типу, або строкових для гіперпараметрів типу вибору). Відповідь на такий запит є точкою у просторі гіперпараметрів, з усіма обмеженнями, що були накладені під час створення експерименту з ідентифікатором <experiment_id>.

За адресою /experiment/<experiment_id>/tell сервіс приймає POST запити у форматі, як зазначено на рисунку 5.5.

```
{
  "point": {
    <param_name>: <value>,
    ...
  },
  "value": <float_number>
}
```

Рисунок 5.5 – Формат запиту для повідомлення сервісу значення цільової функції у певній точці

Як показано на рисунку 5.5, запит має містити координати точки у просторі гіперпараметрів (поле point, формат такий самий як і у попередньому запиті), та значення цільової функції у данній точці (поле value), що має бути числом. Слід зазначити, що такий запит не обов’язково має бути виконаним після отримання зазначеної точки з простору гіперпараметрів за допомогою GET запросу за адресою /experiment/<experiment_id>/ask. У відповідь на успішно оброблений запит сервіс відправить повідомлення { "status": "OK" }.

5.3 WEB-інтерфейс

Для більшої зручності роботи з сервісом було створено веб-сторінку з відносною адресою /dashboard для відображення переліку усіх активних експериментів. На цій сторінці демонструються посилання на візуалізацію для кожного окремого експерименту, а також тут виводяться дані про тип використаного оптимізатору (див. рис. 5.6).

Рисунок 5.6 – Перелік експериментів

Посилання з описаної сторінки ведуть на сторінки для візуалізації процесу оптимізації гіперпараметрів у кожному конкретному експерименті. Шлях до цих

сторінок виглядає так: /dashboard/<experiment_id>, де замість <experiment_id> стоїть ідентифікатор експерименту.

Сторінка експерименту організована наступним чином: на сторінці розташовані графіки, які демонструють обрані на кожній ітерації експерименту гіперпараметри та значення цільової функції. Першим на сторінці відображається графік зміни цільової функції на кожній ітерації експерименту. На цьому графіку на горизонтальній осі відкладені порядкові номери ітерації (починаючи з нуля), а на вертикальній осі – значення цільової функції (див. рис. 5.7).

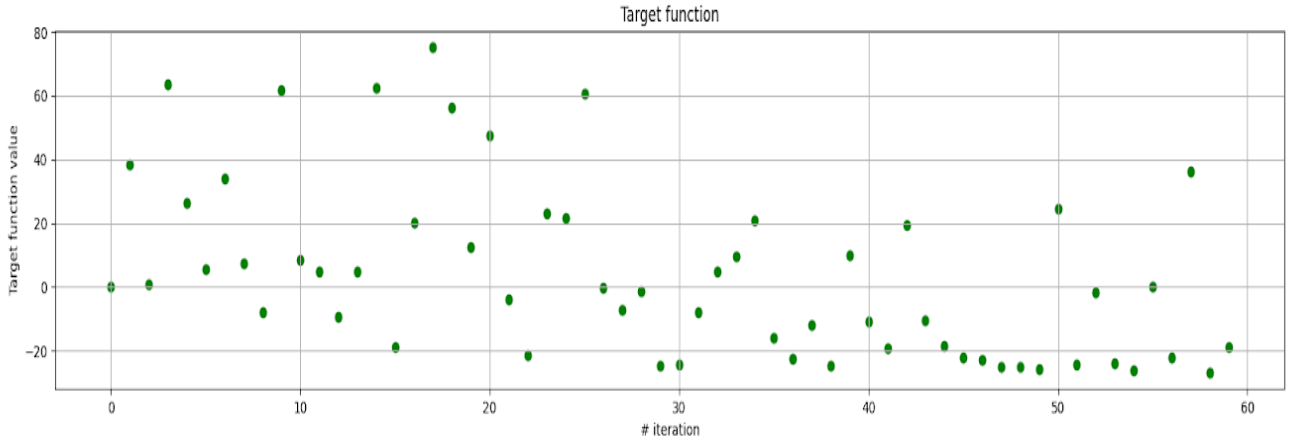

Рисунок 5.7 – Графік цільової функції

Як видно на рисунку 5.7, графік є точковою діаграмою, де кожній ітерації експерименту відповідає одна і тільки одна точка. Відповідно до наведеного графіку, у процесі оптимізації значення гіперпараметрів збігаються до оптимальних значень (що може бути локальним оптимумом).

За цим графіком йдуть графіки для кожного гіперпараметра. Вони побудовані за одним і тим самим принципом: на горизонтальній осі відкладені порядкові номери ітерації, а на вертикальній осі – значення гіперпараметру. Графіки пошуку гіперпараметрів наведені на рисунку 5.8. Точки на таких графіках мають колір в залежності від значення цільової функціїї на даній ітерації. Палітра кольорів та відповідних значень наведена праворуч від графіка.

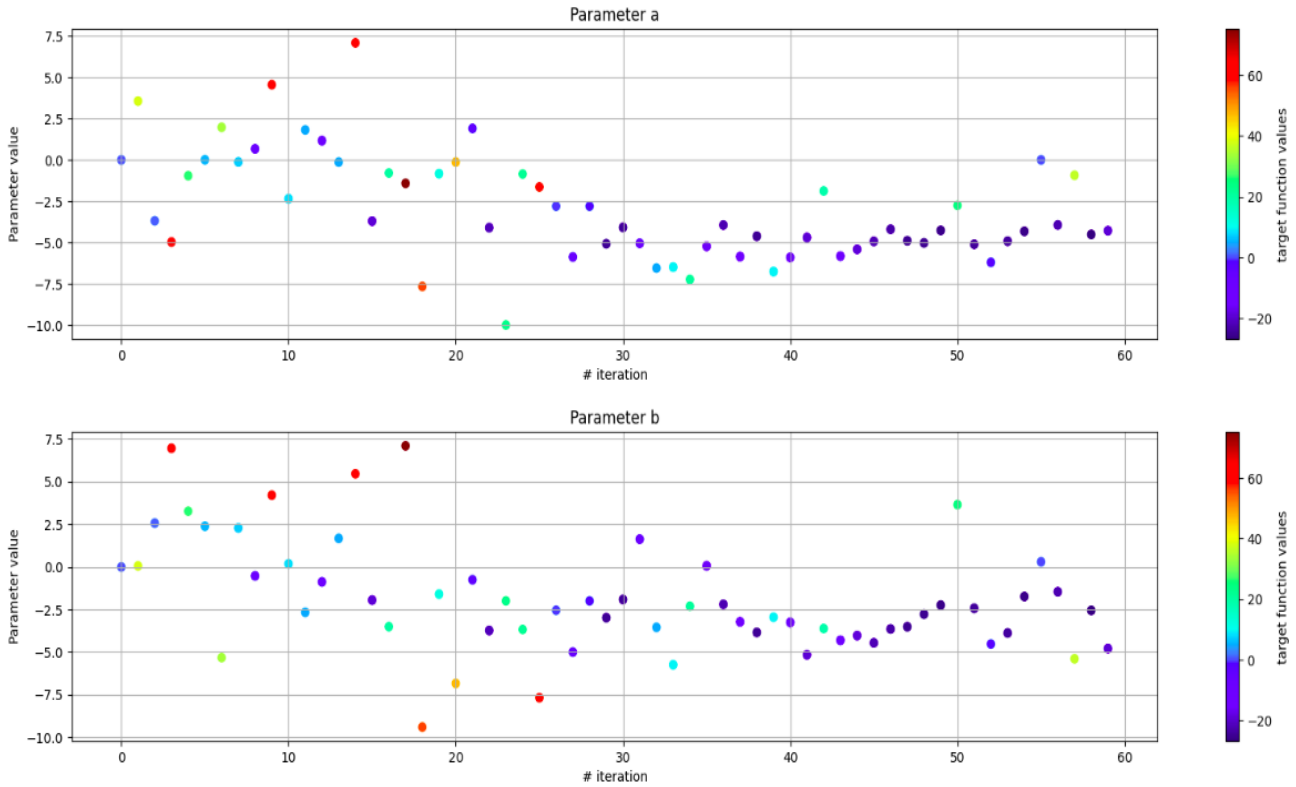

Рисунок 5.8 – Графік значень гіперпараметрів a, b

Для гіперпараметрів типу “choice” значення на вертикальній осі будуть дискретними (рисунок 5.9).

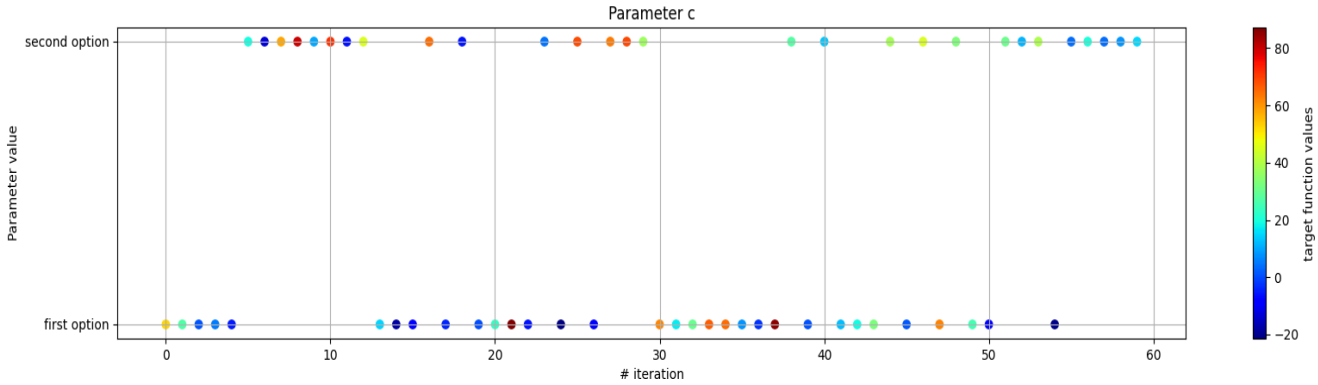

Рисунок 5.9 – Графік значень гіперпараметра типу “choice”

Як видно на попередніх рисунках, графіки до кожного гіперпараметру йдуть один за одним, з однаковою горизонтальною шкалою, що полегшує порівняння ітерацій оптимізації між собою.

5.4 Тестування сервісу у середовищі Jupyter Notebook

Репозиторій проекту на сервісі GitHub включає у себе ноутбуки TestApp.ipynb та SklearnExample.ipynb (див. рис. 5.10). Перший ноутбук призначений для перевірки працездатності розгорнутого сервісу, у той час як другий демонструє використання сервісу для задачі мінімізації похибок моделі машинного навчання – класифікатору побудованого на методі опорних векторів – на прикладі класифікації рукописних цифр.

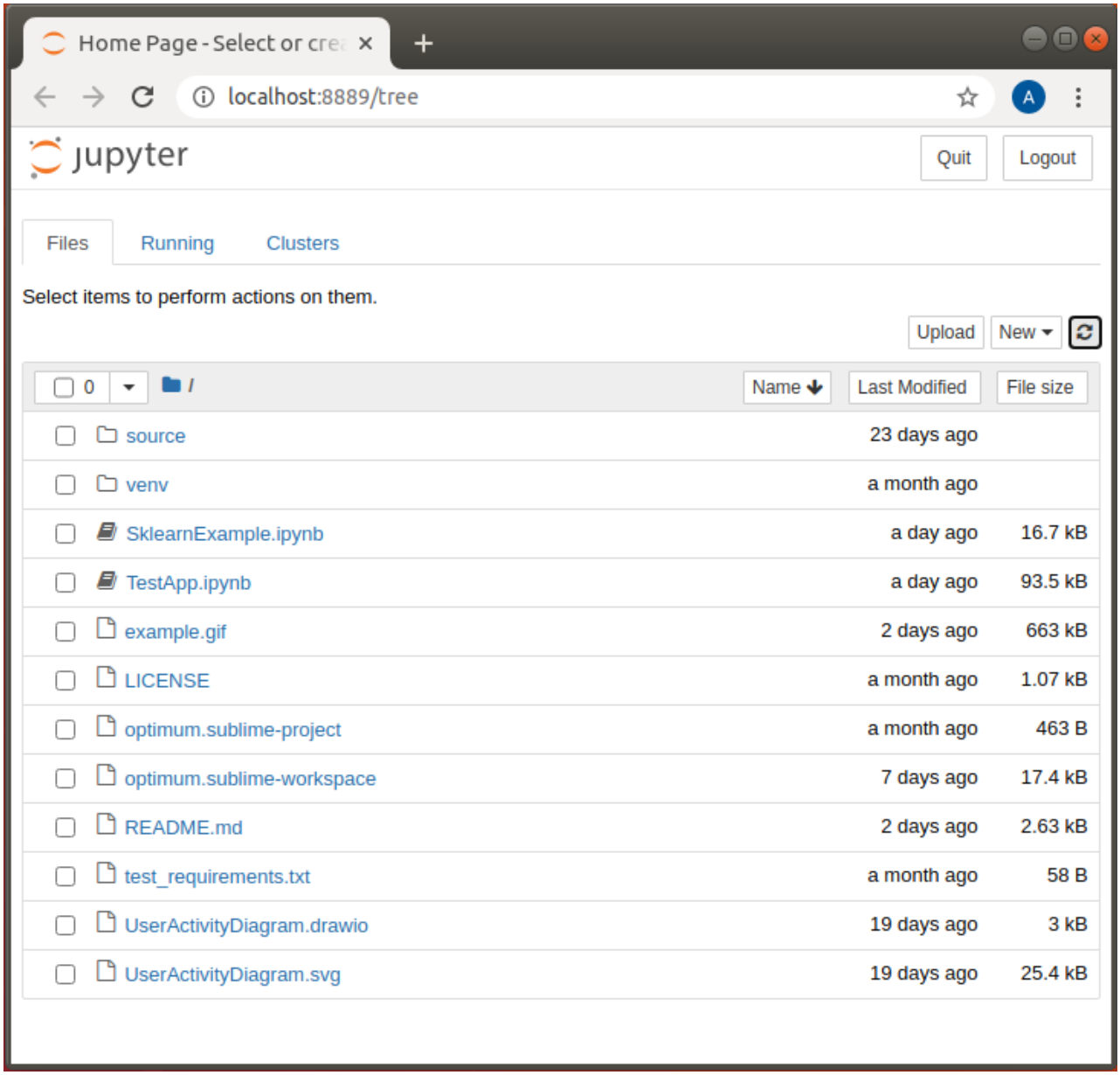

Рисунок 5.10 – Інтерфейс системи jupyter

Дані ноутбуки не є необхідними для функціонування сервісу, проте є зручним інструментом демонстрації користувачеві принципів взаємодії з даним сервісом. Також за допомогою цих ноутбуків можна перевірити працездатність розгорнутого сервісу при виникненні проблем зі взаємодією через користувацькі скрипти.

Ноутбук TestApp.ipynb починається коміркою з імпортом необхідних бібліотек. Далі йдуть 7 комірок з виконанням різних запитів до прикладного програмного інтерфейсу сервісу та виводяться результати надіслані сервісом (див. рис. 5.11).

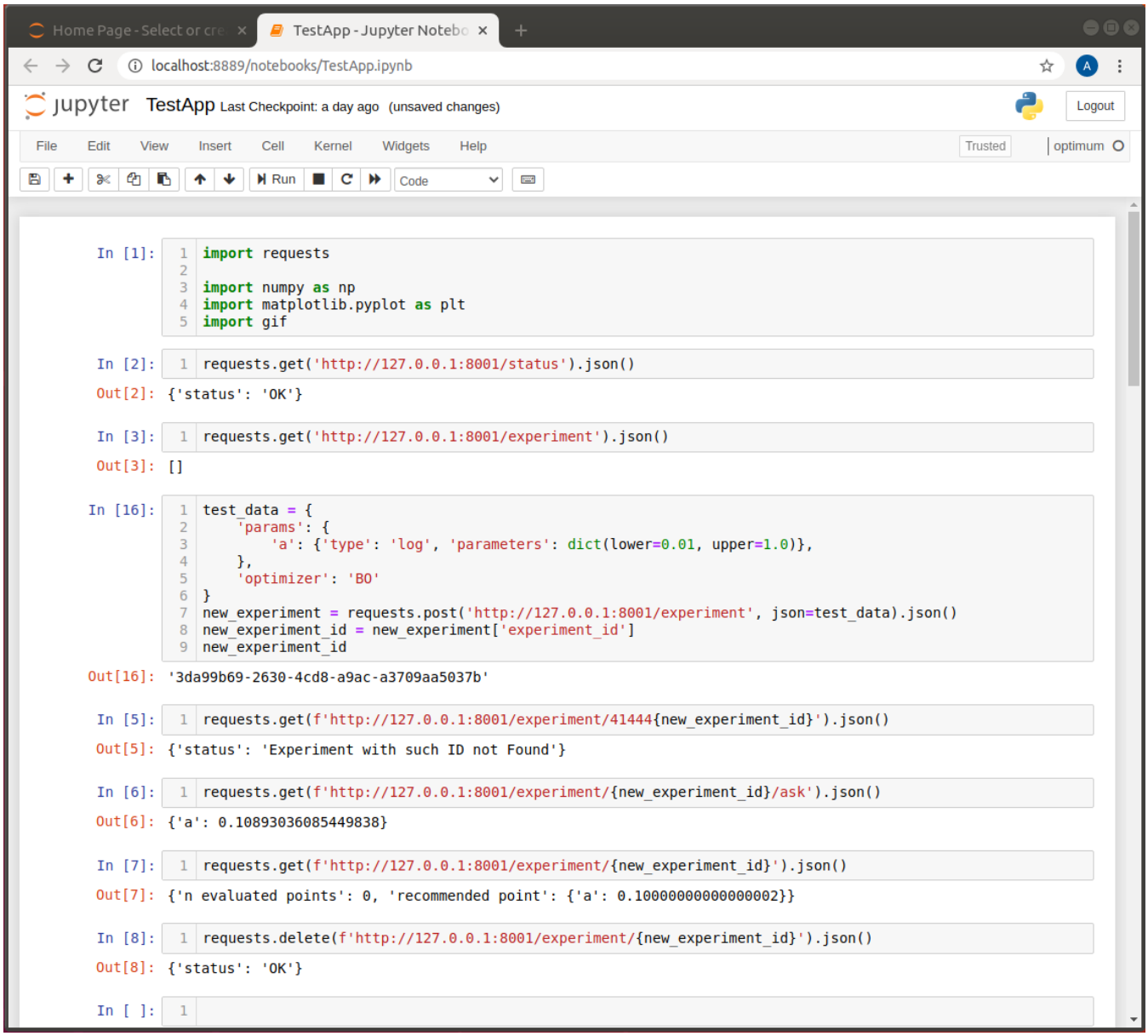

Рисунок 5.11 – Ноутбук TestApp з базовими запитами до локально розгорнутого сервісу

Запуск цих комірок у порядку слідування по одному разу продемонструє відповіді сервісу з даних адрес у штатному режимі (всі запити успішні), повторний запуск довільних комірок у іншому порядку може продемонструвати поведінку сервісу у різних станах, що може бути корисним якщо користувач хоче пересвідчитись у правильності обробки помилок у своєму коді по взаємодії з сервісом. Також, запускаючи наведені комірки у довільному порядку можна відтворити різні стани сервісу у разі необхідності ручного втручання у робочий потік запущеного сервісу (наприклад, вручну видалити певні експерименти, тощо).

Далі у цьому ноутбуці слідує визначення простої функції двох змінних, що має кілька локальних мінімумів, візуалізація значень цієї функції на прямокутній області за допомогою ліній рівня (див. рис. 5.12).

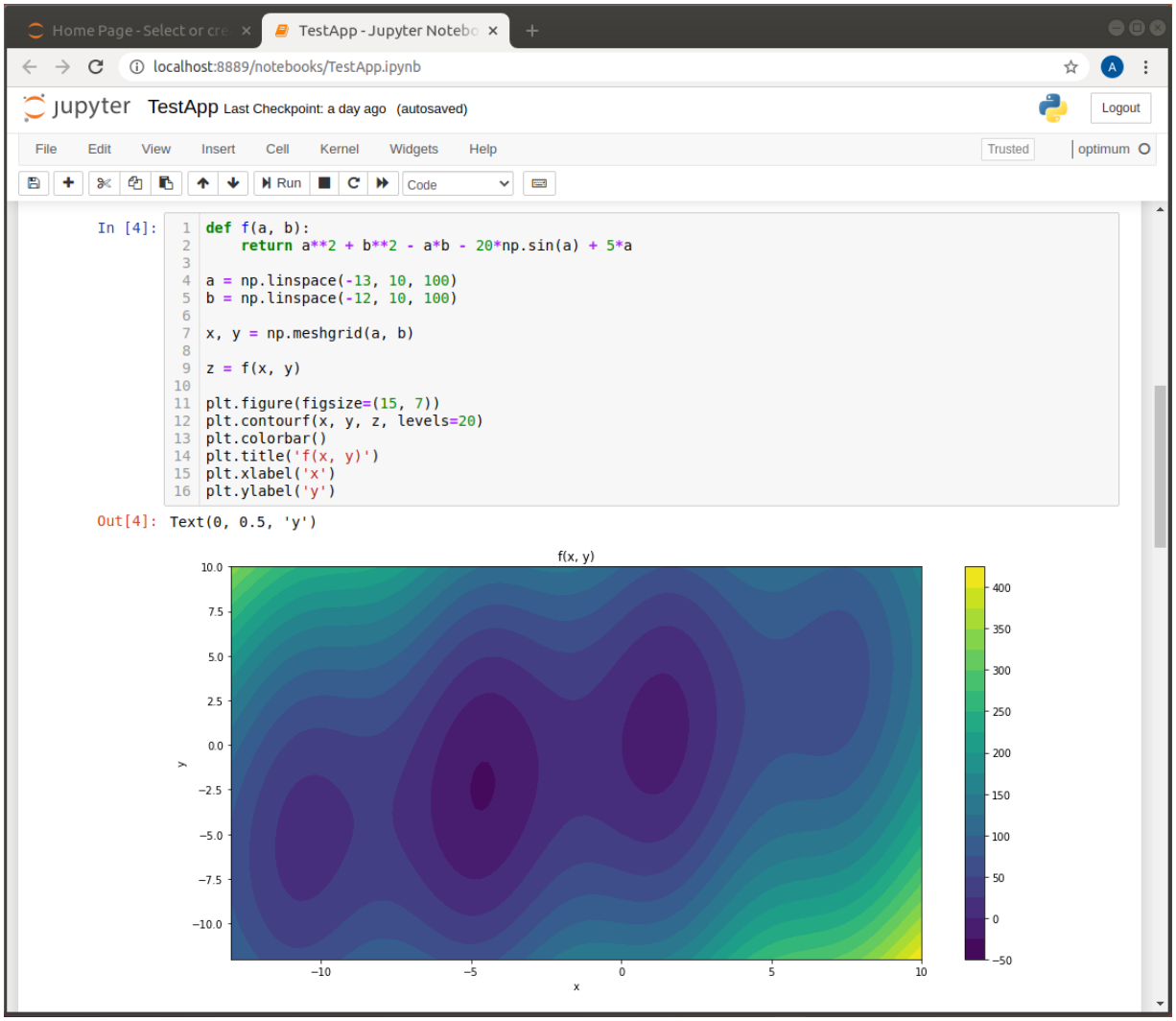

Рисунок 5.12 – Тестова функція двох змінних

Візуалізація виконується засобами бібліотеки matplotlib, результати візуалізації за допомогою цієї бібліотеки за замовчуванням підтримуються та відображаються у Jupyter Notebook.

Після комірки з оголошенням та візуалізацією функції йде комірка з демонстрацією оптимізації цієї функції за допомогою створеного сервісу. У цій комірці створюється новий експеримент, з зазначенням області оптимізації двох параметрів, обмежених максимальним та мінімальним значеннями. Потім у циклі проводиться кілька ітерацій оптимізації, причому на кожній ітерації до сервісу відправляється запит точки з простору параметрів, обчислюється значення раніше оголошеної функції у даній точці, отримане значення відправляється до сервісу для подальшої оптимізації. Також на кожному кроці циклу створюється візуалізація точок, у яких цільова функція вже обчислена, з виділенням останньої точки іншим кольором (див. рис. 5.13). При переході до наступної ітерації поле виводу даної комірки очищується та новий крок візуалізації виводиться на екран. Такий код ефективно призводить до відтворення анімації пошуку оптимального значення цільової функції.

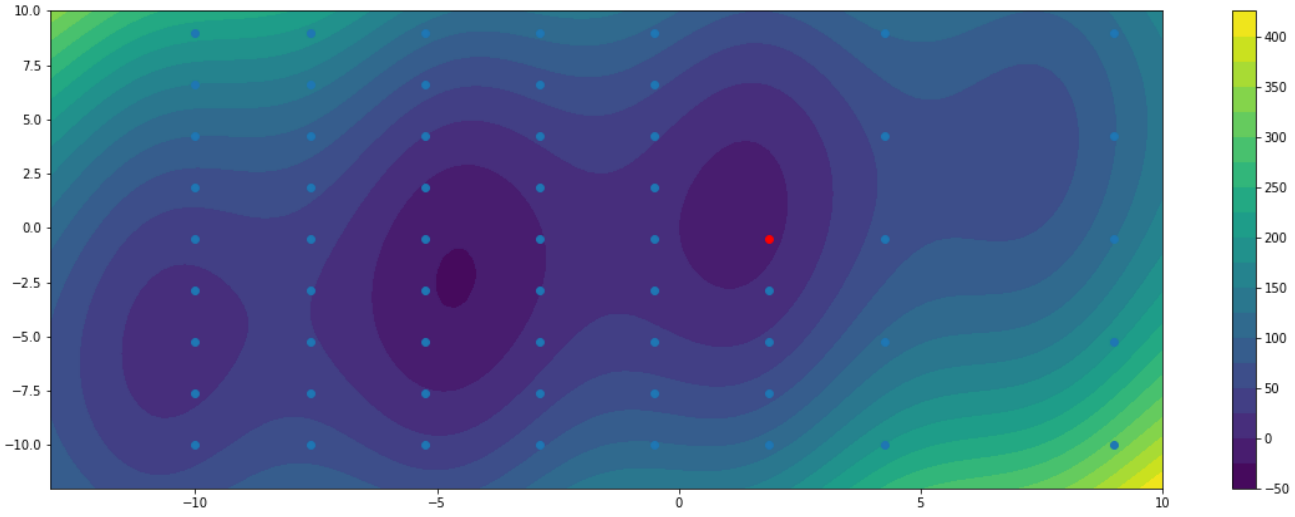

Рисунок 5.13 – Візуалізація перевірених точок у просторі тестової функції

Наприкінці виконання комірки з оптимізацією тестової функції кадри усіх кроків оптимізації перетворюються на анімацію у .gif форматі та записуються на диск користувача у кореневу папку, з якої було запущено Jupyter Notebook.

Наприкінці цього ноутбуку є також комірки у яких виводяться ідентифікатор останнього створеного експерименту та його стан. У повідомленні про стан експерименту наводиться інформація про кількість точок, де було обчислено значення цільової функції та точка, що рекомендована оптимізатором як оптимальна, тобто точка де повідомлене значення цільової функції було мінімальним. У останній комірці ноутбуку виконується обчислення тестової функції у рекомендованій оптимальній точці.

Ноутбук SklearnExample.ipynb демонструє користувачам використання сервісу для мінімізації похибок класифікатора, реалізованого на базі методу опорних векторів. У демонстрації використовується імплементація класифікатора з бібліотеки scikit-learn.

Також з бібліотеки scikit-learn береться вбудований набір даних digits, що є копією набору даних з UCI Machine Learning Repository [19]. Набір даних містить зображення рукописних цифр: 10 класів, де кожен клас посилається на цифру. Програми попередньої обробки, опубліковані NIST, використовувались для отримання нормалізованих растрових зображень рукописних цифр із попередньо роздрукованої форми. Із загальної кількості 43 людей 30 принесли свій внесок у навчальний набір, а 13 – у тестовий набір. Растрові зображення 32 на 32 пікселя поділялись на блоки 4 на 4, що не перекриваються, і кількість пікселів була підрахована у кожному блоці. Це згенерувало матриці 8 на 8, де кожен елемент є цілим числом у діапазоні від 0 до 16. Такі перетворення зменшують розмірність і нівелюють невеликі викривлення [20]. Використаний набір даних містить 1797 записів, кожний запис представлений 64 цілими числами від 0 до 16 включно, кожному запису поставлена у відповідність цифра від 0 до 9, що є цільовим класом для даного запису. В даному прикладі вирішується така задача машинного навчання як класифікація. Так як набір даних є збалансованим (кількість записів що відповідають окремим класам приблизно однакова), то за метріку оцінки роботи класифікатора було обрано середню точність класифікації (тобто долю записів з певного оціночного набору даних, у яких клас передбачений моделлю співпадає з істинним класом даного запису). Відповідно, за цільову функцію для мінімізації було

обрано долю похибок, тобто неправильно класифікованих записів з тестового набору даних.

У першій комірці ноутбуку імпортуються необхідні бібліотеки та функції, далі виконується загрузка набору даних та його розподіл на записи та масив з цільовими класами цих записів.

У наступній комірці ноутбуку створюється новий експеримент, з простором оптимізації з трьома гіперпараметрами для класифікації за методом опорних векторів:

- гіперпараметер kernel типу вибору, який відповідає ядру, що використовується для методу опорних векторів у класифікаторі. Обирається з наступних варіантів: 'linear', 'poly', 'rbf', 'sigmoid', тобто лінійне ядро, поліноміальне, RBF, сигмоідальне;
- гіперпараметер C, що є константою регуляризації. Для нього встановлено числовий тип зі значеннями від 0.01 до 1. Такий вибір значень обумовлений тим, що у даній реалізації методу опорних векторів цей гіперпараметер обернено пропорційний до сили регуляризацій, та має бути строго більше нуля. 0.01 було встановлено як мінімальне значення для цього параметру, бо меньші значення призводять до дуже значної регуляризації значень параметрів моделі, що веде до загального збільшення похибки моделі. Значення більше 1 можуть призводити до стрімкого росту параметрів моделі, що призведе до “перенавчання”, коли модель навчається параметрам, що занадто упередженні до тренувального набору даних і як результат дають більше помилок на тестовому наборі;
- гіперпараметер degree числового типу, що є найвищою степінню поліномів для використання у поліноміальному ядрі методу опорних векторів. Цей гіперпараметер не впливає на модель машинного навчання при використанні інших ядер, окрім поліноміального.

Після створення експерименту з наведеними обмеженнями на область гіперпараметрів виконується цикл ітерацій оптимізації. На кожній ітерації у поле виводу комірки виводиться інформація про поточне значення цільової функції.

Значення цільової функції також було збережено у окремий список для подальшої візуалізації.

За результатами мінімізації похибки класифікатора шляхом проведення 25 ітерацій оптимізації (ітерації індексуються від 0 до 24) виявлено значення гіперпараметрів, що зменьшують значеня похибки моделі приблизно у півтори рази. Значення похибки, що були мінімальними після проведення певної кількості ітерацій наведено у таблиці 5.1.

Таблиця 5.1 – Результати обчислень

Номер ітерації	0	4	8	12	16	20	24
Похибка моделі	0.0512	0.0356	0.0356	0.0356	0.0356	0.0339	0.0339

Дані про похибку на кожному кроці ітерації, а також значення мінімальної відомої похибки на данном кроці і найменьше знайдене значення продемонстровано на рисунку 5.14.

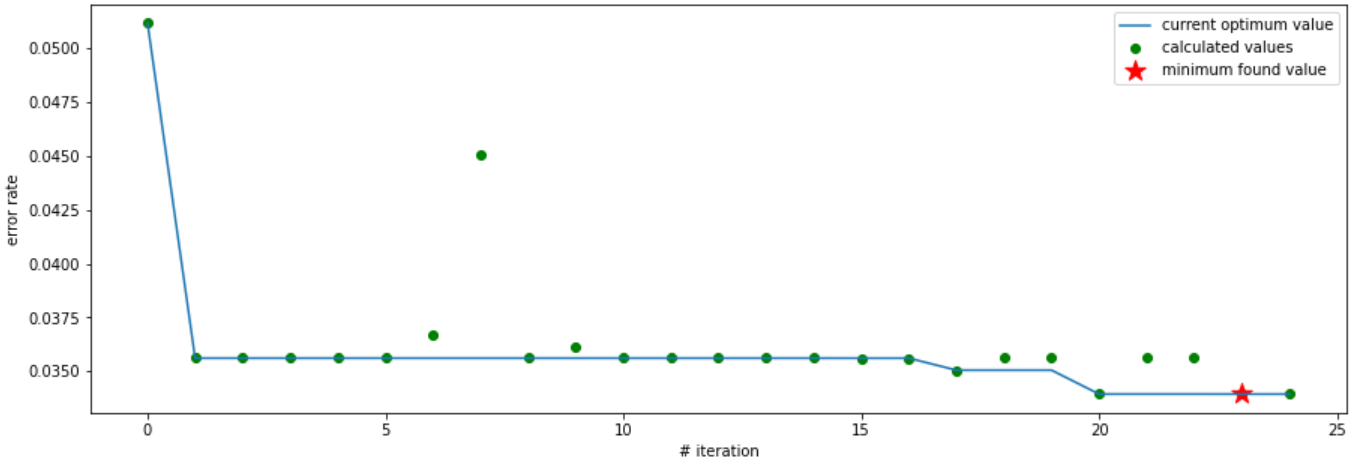

Рисунок 5.14 – Мінімізація частоти похибок класифікатора зображень за допомогою байєсівської оптимізації

На рисунку 5.14 точками зображено значення цільової функції (частоти похибок класифікатора), блакитною лінією позначено мінімальне відоме значення цільової функції на певній ітерації, червоною зіркою – знайдене мінімальне значення.

6 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблений сервіс насамперед призначений для полегшення та автоматизації процесу оптимізації гіперпараметрів моделей машинного навчання. Найпростішим способом запустити сервіс є використання публічного образу з Docker Hub. Для цього треба мати встановлений docker (інструкції по інсталяції можна знайти на офіційному сайті). Далі для локального запуску сервісу достатньо виконати команду “docker run -p 8001:8001 alex314mart/optimum”. Образ загрузиться на машину користувача, буде створено контейнер і в ньому запущено сервіс. Також дана команда перенаправляє повідомлення з локального порту 8001 на відповідний порт контейнеру, для взаємодії з сервісом. У разі виникнення конфліктів локальний порт для взаємодії з сервісом можна змінити, детальніше див. офіційну документацію щодо команди docker run та флага -p.

Після запуску сервісу можливість взаємодії можна перевірити відправивши GET-запит за відносною адресою /status (для нашого прикладу на локальній машині - 127.0.0.1:8001/status). Цей запит поверне JSON-відповідь зі статусом (див. рис. 6.1).

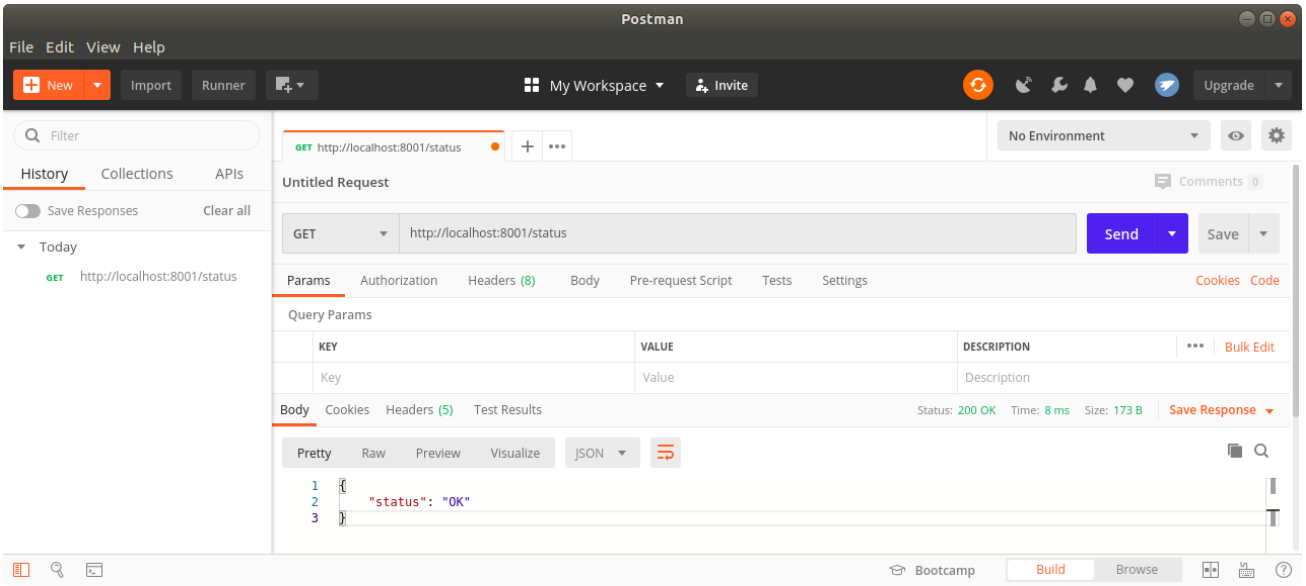

Рисунок 6.1 – Запит статусу сервісу

Запити статусу можна подалі використовувати для контролю автоматичного розгортання сервісу. Наприклад, широко відомою практикою є пробування сервісів на працездатність у системі Kubernetes (так звані liveness probe) що може бути реалізовано за допомогою таких запитів статусу. Тут і надалі для демонстрування роботи API використано програмний продукт Postman, що є популярним інструментом для работи з різними API.

Після успішного запуску сервісу можна переходити до створення експерименту, та спочатку треба обрати приклад задачі. Так як задачу оптимізації гіперпараметрів можна представити як задачу оптимізації функції багатьох змінних, розглянемо на прикладі оптимізацію функції двох змінних. Такий приклад полегшить візуалізацію задачі, процесу розв’язання та отриманих результатів і буде більш наочно демонструвати можливості сервісу. Візьмемо для прикладу функцію, яка має локальні мінімуми і візуалізуємо її лініями рівня (див. рис. 6.2).

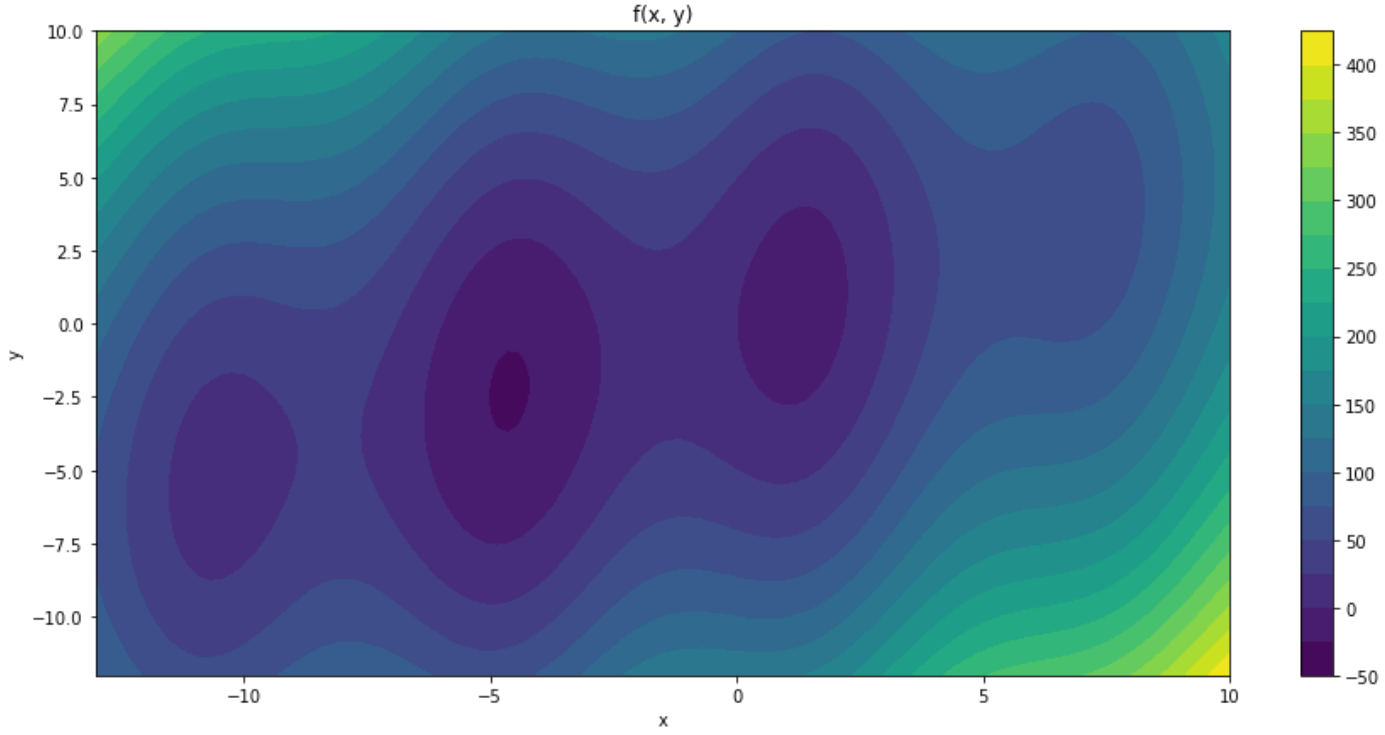

Рисунок 6.2 – Візуалізація тестової функції

По-перше, для використання сервісу необхідно створити новий експеримент. Для цього необхідно обрати оптимізатор. Оберемо Байєсівську оптимізацію, назва

оптимізатора – “BO” від Bayesian optimization. Також необхідно зазначити простір параметрів. Для цього у запиті передаються назви параметрів (в наведеному прикладі a, b), тип кожного параметру (“scalar” відповідає числовому типу) і додаткові обмеження. Обидва параметри обмежено від -10 до 10. Всю цю інформацію формуємо у JSON та відправляємо POST-запитом за адресою /experiment (127.0.0.1:8001/experiment для локальної машини) згідно з прийнятим форматом передачі даних (див. рис. 6.3).

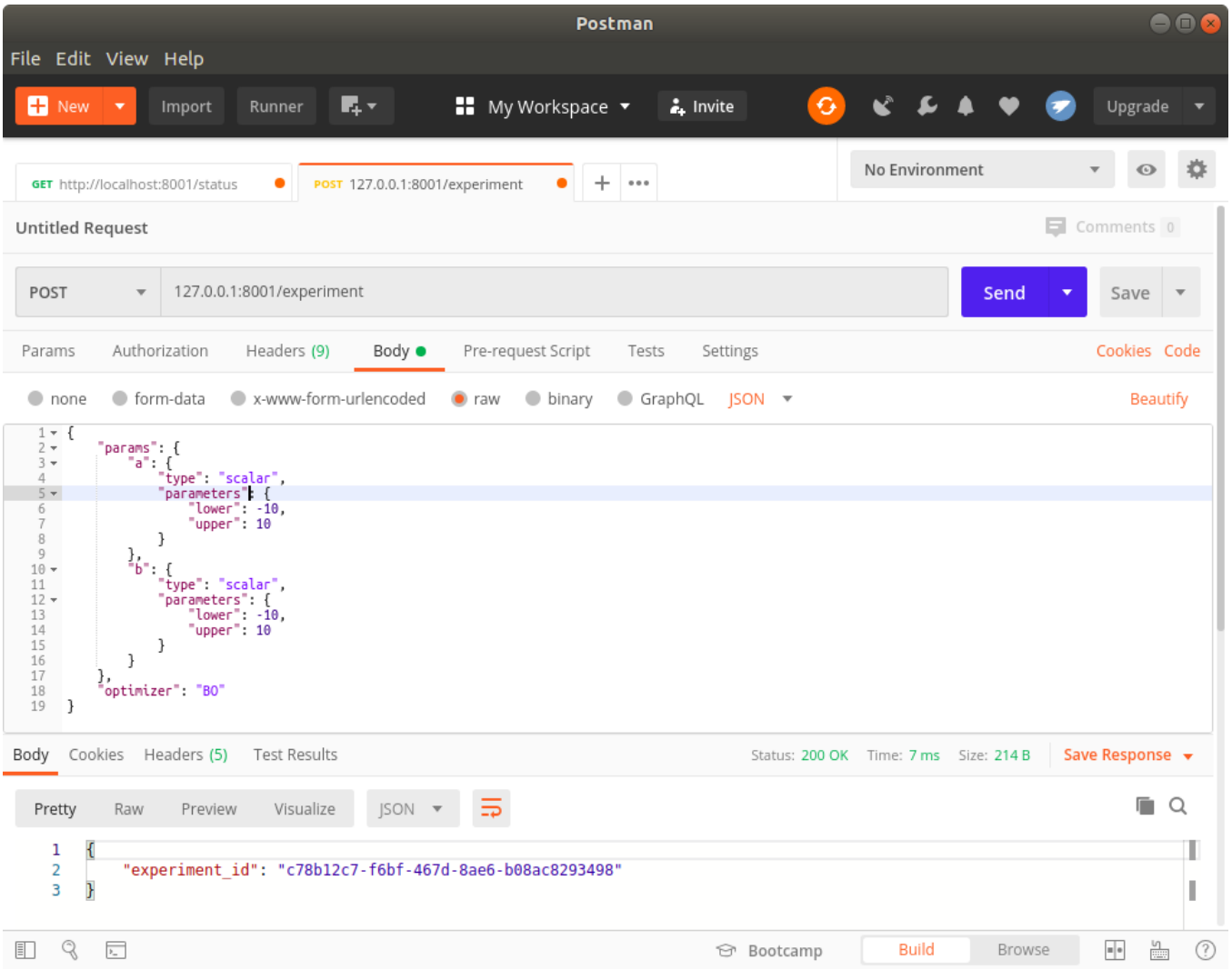

Рисунок 6.3 – Запит на створення експерименту

У відповідь на запит отримаємо JSON з ідентифікатором створеного експерименту. Подальша робота з експериментом реалізується за схемою “спитати - повідомити”. Клієнт запитує у сервісу, яку точку у просторі параметрів треба

перевірити, обчислює значення цільової функції у заданій точці та повідомляє результат сервісу.

Для запиту рекомендованої точки треба відправити GET-запит за адресою /experiment/<experiment_id>/ask, де <experiment_id> - це ідентифікатор експерименту, отриманий при його створенні. У відповідь отримаємо JSON з даними у форматі параметр - значення (див. рис. 6.4). Для наведеного прикладу повна адреса запиту виглядатиме так: 127.0.0.1:8001/experiment/c78b12c7-f6bf-467d-8ae6-b08ac8293498/ask

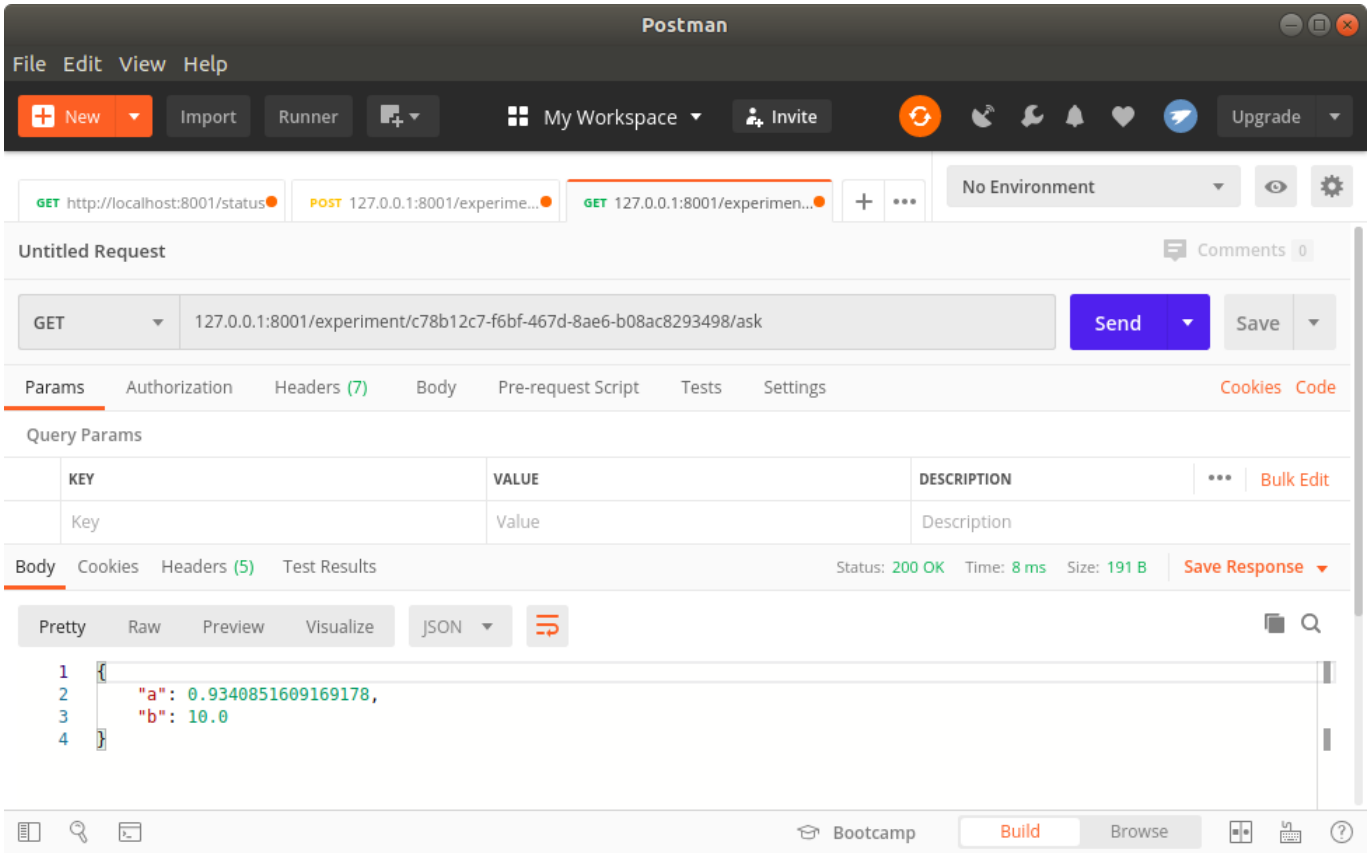

Рисунок 6.4 – Запит точки для обчислення цільової функції

Розрахуємо значення цільової функції та передамо його сервісу. Для цього відправимо POST-запит за адресою /experiment/<experiment_id>/tell. Повна адреса для наведеного прикладу: 127.0.0.1:8001/experiment/c78b12c7-f6bf-467d-8ae6-b08ac8293498/tell

У тіло запиту запишемо точку у якій проводили обчислення та, власне, значення цільової функції. У відповіді буде підтвердження в вигляді статусу OK (див. рис. 6.5).

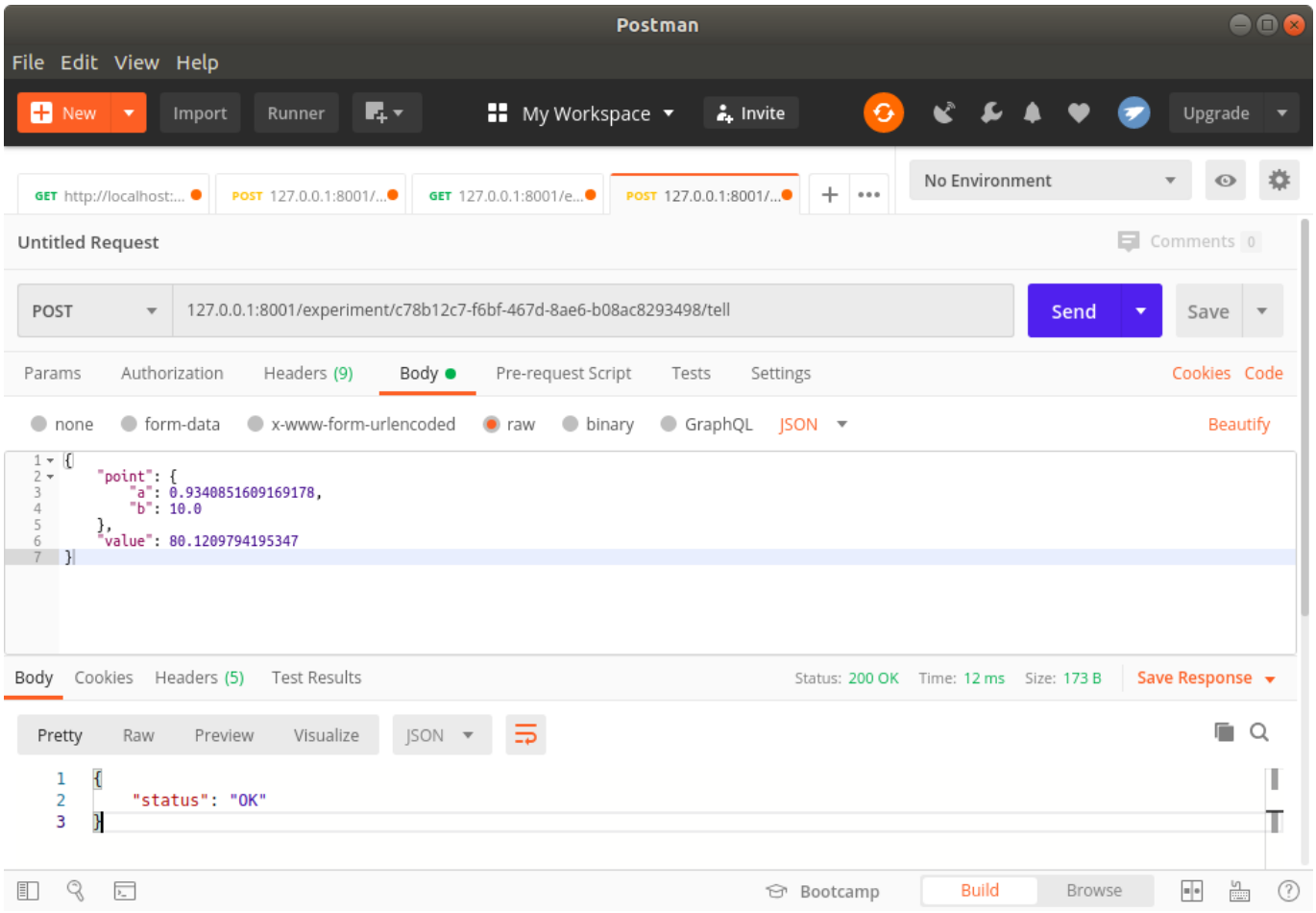

Рисунок 6.5 – Повідомлення значення цільової функції

Проведемо 60 ітерацій обчислення цільової функції у рекомендованих точках. За прогресом можна слідкувати через web-інтерфейс. Зокрема, у переліку активних експериментів за адресою /dashboard буде запис з ідентифікатором створеного експерименту та зазначеним оптимізатором, що використовується. Клік на цей запис перейде за посиланням /dashboard/<experiment_id>. Для проведеного тесту це буде 127.0.0.1:8001/dashboard/c78b12c7-f6bf-467d-8ae6-b08ac8293498

За зазначеною адресою побачимо три графіки: графік значень цільової функції по ітераціям та два графіки значень параметрів a, b. Бачимо, що Байєсівська оптимізація збігається до мінімуму, інколи перевіряючи точки зі значеннями

параметрів що істотно відрізняються від значень біля мінімуму цільової функції. Це робиться щоб у разі потрапляння у локальний мінімум мати змогу знайти глобальний мінімум цільової функції у заданому просторі (див. рис. 6.6).

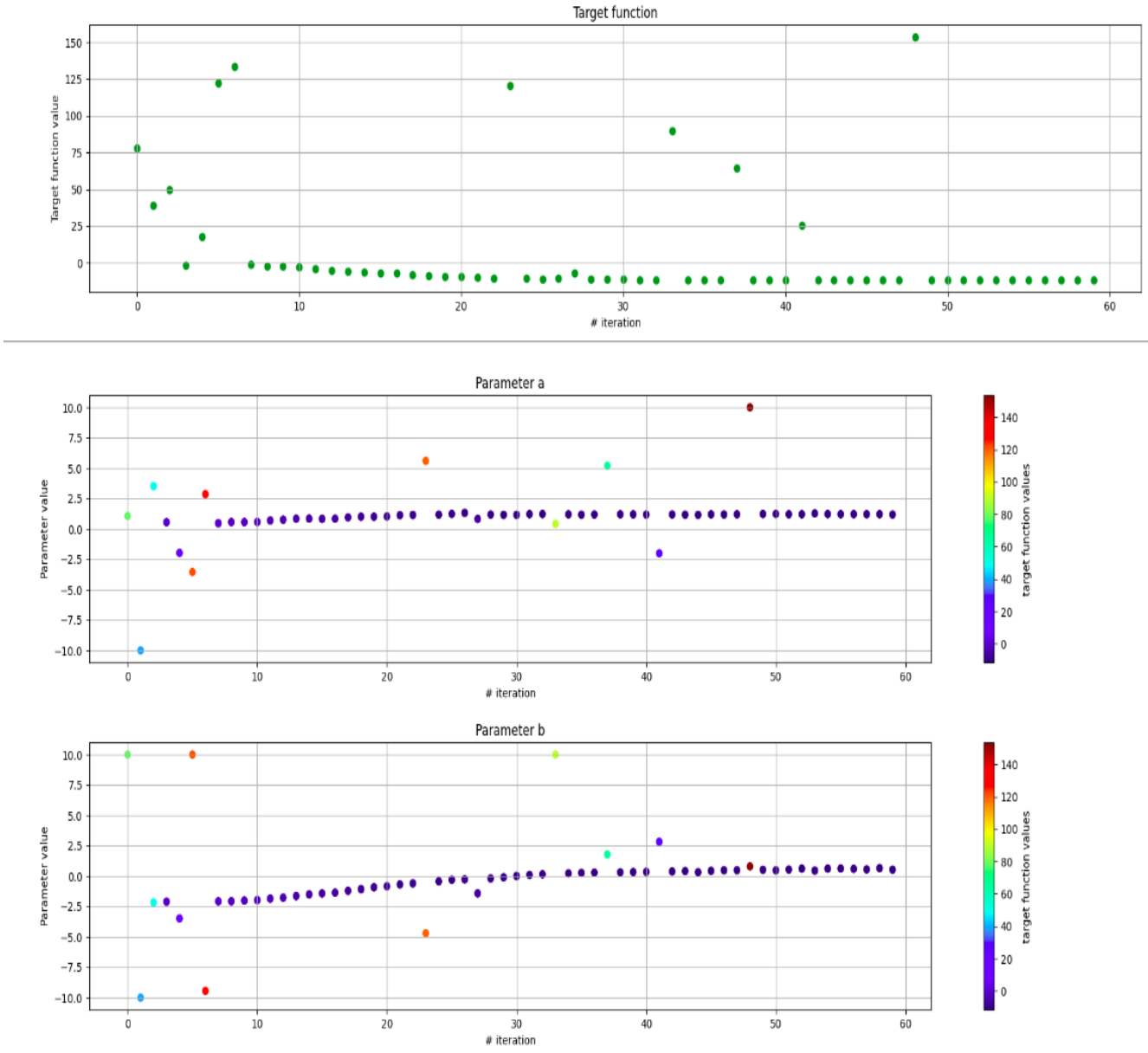

Рисунок 6.6 – Візуалізація процесу оптимізації

Задля отримання рекомендованого оптимізатором значення параметрів, тобто такого, при якому значення цільової функції мінімальне, треба зробити GET-запит за адресою /experiment/<experiment_id>. Також, такий запит надасть інформацію про кількість проведених обчислень цільової функції (див. рис. 6.7).

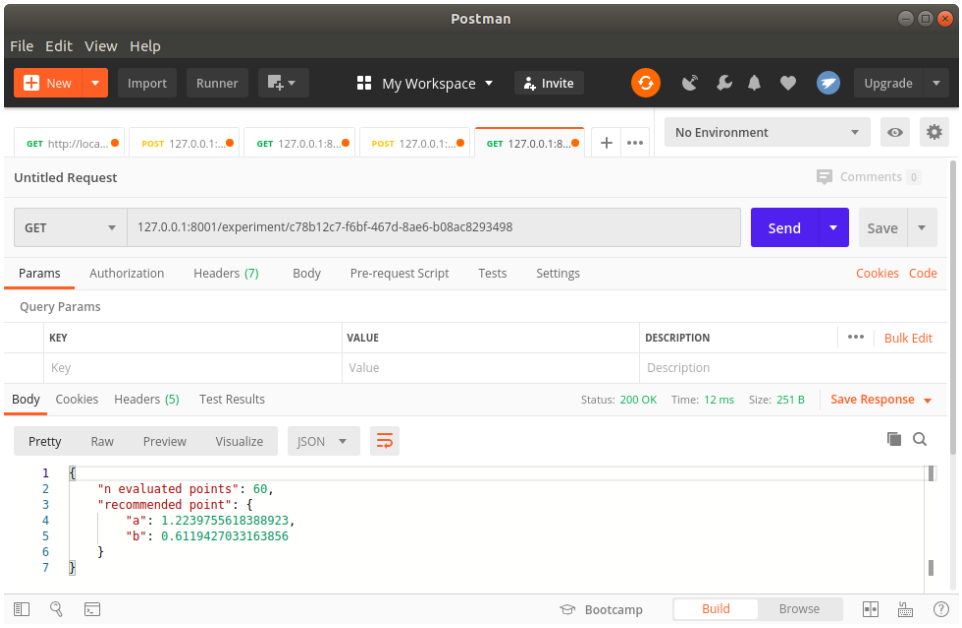

Рисунок 6.7 – Запит статусу певного експерименту

У сервісі реалізована можливість розширення функціоналу через додавання користувачем власних алгоритмів оптимізації. Для цього наведено приклад імплементації пошуку за сіткою зі змінним кроком (див. рис. 6.8)

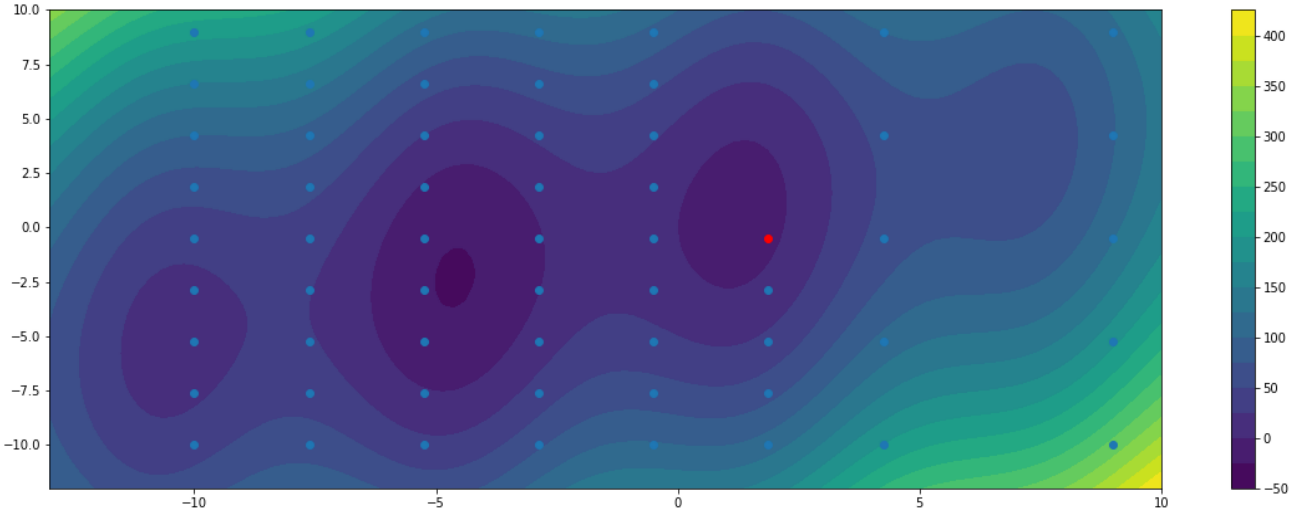

Рисунок 6.8 – Візуалізація перевірених точок за допомогою демонстраційного оптимізатору GridSearch

ВИСНОВКИ

В ході виконання даної роботи було отримано наступні результати:

1. Проведено огляд існуючих програмних рішень в області мінімізації похибок моделей машинного навчання шляхом пошуку оптимальних гіперпараметрів, який показав актуальність створення нового програмного продукту, що забезпечує безпеку даних шляхом розгортання сервісу на стороні клієнта, залишаючи такі переваги існуючих платформ, як незалежність від особливостей клієнтської реалізації процесу тренування моделей машинного навчання.
2. Проведено огляд існуючих алгоритмів пошуку отптимальних гіперпараметрів у багатовимірному просторі і для застосування при розробці сервісу обрано методи байєсівської оптимізації, генетичні алгоритми.
3. Вибрано засоби розробки сервісу: мова програмування Python через сильну динамічну типізацію та широкий набір стандартних бібліотек; систему контроля версій git та сервіс GitHub задля полегшення подальшого вдосконалення сервісу та через популярність серед потенційних клієнтів; сервіс Docker Hub задля полегшення розповсюдження створеного сервісу; середовище jupyter notebook задля наочної демонстрації можливостей продукту та для полегшення мануального тестування.
4. Розроблено архітектурні рішення побудови сервісу та інтерфейсу його взаємодії з користувачем. Для реалізації прикладного програмного інтерфейсу було обрано передачу повідомлень у форматі JSON через невелике додаткове навантаження формату у порівнянні з розміром повідомлення, збереження читаємості повідомленнь людиною та загального поширення формату і інстументів роботи з ним.
5. Проведено мануальне тестування сервісу, продемонстровані тестові приклади використання окремих функцій сервісу на прикладі оптимізації аналітично заданої функції двох зміних, а також продемонстрорвано використання сервісу для мінімізації похибок моделі-класифікатора

зображень. Результати тестування підтвердили працездатність сервісу та відповідність поставленим вимогам.

6. У тексті роботи надано детальний опис інтерфейсу взаємодії користувача з системою, що дозволяє налагодити ефективну взаємодію з сервісом. Наведено приклади запитів до системи та отриманих відповідей.
7. Розроблений сервіс дозволяє створювати експерименти по оптимізації гіперпараметрів моделей машинного навчання, з підтримкою оперування кількома експериментами одночасно. Сервіс надає прикладний програмний інтерфейс для взаємодії з користувацькими програмними засобами, а також веб-інтерфейс для візуального контролю за станом процесу мінімізації похибок моделей машинного навчання. Створений сервіс є незалежним від фреймворку, мови програмування, платформи моделей машинного навчання, що оптимізуються; має можливість розгортання на стороні користувача. У сервісі реалізована можливість розширення функціоналу через додавання користувачем власних алгоритмів оптимізації.

Результати виконаної роботи можуть бути використані фахівцями зі сфери науки про дані під час створення та впровадження моделей машинного навчання у прикладних застосунках. Вихідний код створеного програмного засобу публічно доступний на сервісі GitHub, а docker-образ сервісу публічно доступний у реєстрі образів Docker Hub.

В подальших дослідженнях слід розглянути роботу сервісу у високонагружених паралельних системах, а також збереження та відтворення стану сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M.Feurer. Hyperparameter optimization / M.Feurer, F.Hutter. // Automated Machine Learning. The Springer Series on Challenges in Machine Learning. – Springer, Cham, 2019.
2. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective / Kevin P. Murphy. – MIT Press, 2012.
3. Andreas C. Müller. Introduction to Machine Learning with Python: A Guide for Data Scientists. – Andreas C. Müller, Sarah Guido – O'Reilly Media, 2016.
4. Claesen, M. and B. D. Moor. Hyperparameter search in machine learning. arXiv 1502.02127. 2015.
5. James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization // Journal of Machine Learning Research. 2012. №13(1). С. 281–305.
6. Petro Liashchynskyi, Pavlo Liashchynskyi. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS / 2019.
7. John McCall. Genetic algorithms for modelling and optimization // Journal of Computational and Applied Mathematics. 2005.
8. Snoek, Jasper; Larochelle, Hugo; Adams, Ryan. Practical Bayesian Optimization of Machine Learning Algorithms // Advances in Neural Information Processing Systems. arXiv:1206.2944. 2012.
9. Мартиненко О.П. Оптимізація гіперпараметрів моделей машинного навчання / Мартиненко О.П., Шушура О.М. // VIII Всеукраїнська науково-практична конференція з автоматичного управління — Херсон, 2020.
10. Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools [Електронний ресурс] // Medium. 2018. — Режим доступу: https: towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a .
11. James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms // Proceedings of the 12th Python in Science Conference. 2013. С. 13–20.

12. Документація kubeflow/Katib [Електронний ресурс] — Режим доступу: https://www.kubeflow.org/docs/components/hyperparameter-tuning/

13. Python Cookbook: Recipes for Mastering Python 3. - David Beazley, Brian K. Jones - O'Reilly Media, 2013.

14. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming. - O'Reilly, 2014.

15. Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. - O'Reilly, 2014.

16. S. Newman. Building microservices. - O'Reilly, 2015.

17. Pro Git 2nd Edition [Електронний ресурс] / Scott Chacon, Ben Straub — 2014 — Режим доступу: https://git-scm.com/book/en/v2 .

18. Документація Docker Hub [Електронний ресурс] — Режим доступу: https://docs.docker.com/docker-hub/ .

19. Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

20. M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.

Додаток А

Технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів

Специфікація

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б

Аркушів 1

Київ — 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б	Пояснювальна записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б 12-1	app.py, hyperApp.py, optimizers.py	Основний функціонал та реєстр оптимізаторів
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б 13-1	Додаток В.docx	Опис програмних модулів

Додаток Б

Технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів

Лістинг програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б 12-1

Аркушів 9

Київ — 2020

app.py

```
import logging

from flask import request, jsonify, render_template

from hyperApp import HyperApp


app = HyperApp(__name__)
app.logger.setLevel(logging.INFO)


@app.route("/status", methods=['GET'])
def status():
    """Status of service"""
    app.logger.info("Status requested")
    return {'status': 'OK'}


@app.route("/experiment", methods=['GET', 'POST'])
def experiment():
    if request.method == 'POST':  # Create new experiment
        params = request.json
        app.logger.info(f'Request experiment creation: {params}')
        experiment_id = app.start_experiment(params)
        app.logger.info(f'Experiment created: {experiment_id}')
        return {'experiment_id': experiment_id}
    else:  # List all active experiments
        ans = [{'experiment_id': experiment_id} for experiment_id in app.experiments]
        return jsonify(ans)


@app.route("/experiment/<experiment_id>", methods=['GET', 'DELETE'])
def particular_experiment(experiment_id):
    if experiment_id not in app.experiments:
```

```
        return {'status': 'Experiment with such ID not Found'}, 404

    if request.method == 'DELETE':  # Delete experiment
        del app.experiments[experiment_id]
        app.logger.info(f'Experiment {experiment_id} deleted')
        return {'status': 'OK'}
    else:  # Get status and current best point for experiment
        status = app.get_status(experiment_id)
        return status


@app.route("/experiment/<experiment_id>/ask", methods=['GET'])
def ask(experiment_id):
    if experiment_id not in app.experiments:
        return {'status': f'Experiment with ID {experiment_id} not Found'}, 404
    point = app.ask(experiment_id)
    app.logger.info(f'Point to return: {point}')
    return point


@app.route("/experiment/<experiment_id>/tell", methods=['POST'])
def tell(experiment_id):
    if experiment_id not in app.experiments:
        return {'status': f'Experiment with ID {experiment_id} not Found'}, 404
    point = request.json['point']
    value = request.json['value']
    app.logger.info(f'Tell value: {value} at point: {point}')
    app.tell(experiment_id, point, value)
    return {'status': 'OK'}


@app.route("/dashboard", methods=['GET'])
def dashboard():
    experiments = [[experiment_id, params['optimizer_title']]
                   for experiment_id, params in app.experiments.items()]
```

```
    return render_template('experiments.html', experiments=experiments)


@app.route("/dashboard/<experiment_id>", methods=['GET'])
def dashboard_experiment(experiment_id):
    app.logger.info(f"Visualization requested for {experiment_id}")
    if experiment_id not in app.experiments:
        return f'Experiment with ID {experiment_id} not Found', 404
    target_chart = app.get_target_chart(experiment_id)
    charts = app.get_charts(experiment_id)
    return render_template('charts.html', target_chart=target_chart, charts=charts)


if __name__ == '__main__':
    app.run({}, host='0.0.0.0', port=5000)
```

hyperApp.py

```
import uuid
from io import BytesIO
import base64

from flask import Flask
import nevergrad as ng
import matplotlib.pyplot as plt

from optimizers import get_optimizer



class HyperApp(Flask):
    def __init__(self, *args, **kwargs):
        super(self.__class__, self).__init__(*args, **kwargs)
        self.experiments = {}

    def run(self, params, *args, **kwargs):
        super().run(*args, **kwargs)

    def parse_params_space(self, params):
        ng_param_types = {
            'log': ng.p.Log,
            'choice': ng.p.Choice,
            'scalar': ng.p.Scalar,
        }
        ng_params = {}
        for parameter_name, specs in params.items():
            if not isinstance(specs, dict):
                raise ValueError(f'Parameter specifications should be dict for {parameter_name!r}')
            param_type = specs.get('type', None)
            if param_type is None:
                raise ValueError(f'No type found for parameter {parameter_name!r}')
            param_attributes = specs.get('parameters', {})
```

```
                try:
                    ng_params[parameter_name] = ng_param_types.get(param_type,
None)(**param_attributes)
                except Exception:
                    raise ValueError(f'Unknown attributes for parameter {parameter_name!r}')
            return ng.p.Dict(**ng_params)

        def start_experiment(self, request):
            params = request.get('params')
            optimizer_title = request.get('optimizer', 'OnePlusOne')
            params_space = self.parse_params_space(params)
            self.logger.info(f'Params space defined: {str(params_space)}')
            optimizer = get_optimizer(optimizer_title)(parametrization=params_space)
            experiment_id = str(uuid.uuid4())
            self.experiments[experiment_id] = {
                'optimizer_title': optimizer_title,
                'optimizer': optimizer
            }
            return experiment_id

        def ask(self, experiment_id):
            optimizer = self.experiments[experiment_id]['optimizer']
            point = optimizer.ask()
            return point.value

        def tell(self, experiment_id, point, value):
            optimizer = self.experiments[experiment_id]['optimizer']
            candidate = optimizer.parametrization.spawn_child(new_value=point)
            optimizer.tell(candidate, value)

        def get_status(self, experiment_id):
            optimizer = self.experiments[experiment_id]['optimizer']
            ans = {
                'n evaluated points': len(optimizer.archive),
                'recommended point': optimizer.recommend().value,
```

```
    }
    return ans

def get_target_chart(self, experiment_id):
    optimizer = self.experiments[experiment_id]['optimizer']
    values = []
    for checked in optimizer.archive.values():
        values.append(checked.mean)

    plt.figure(figsize=(20, 4))
    plt.scatter(range(len(values)), values, c='g')
    plt.title('Target function')
    plt.xlabel('# iteration')
    plt.ylabel('Target function value')
    plt.grid()

    figfile = BytesIO()
    plt.savefig(figfile, format='png')
    figfile.seek(0)
    figdata_png = base64.b64encode(figfile.getvalue()).decode()
    ans = ' ' + figdata_png
    plt.close('all')
    return ans

def get_charts(self, experiment_id):
    optimizer = self.experiments[experiment_id]['optimizer']
    charts = []

    checked_params = {}
    values = []
    for checked in optimizer.archive.values():
        values.append(checked.mean)
        for k, v in checked.parameter.value.items():
            checked_params[k] = checked_params.get(k, []) + [v]
```

```
    for parameter, parameter_values in checked_params.items():
        plt.figure(figsize=(20, 4))
        plt.scatter(range(len(values)), parameter_values, c=values, cmap='jet')
        plt.colorbar(label='target function values')
        plt.title(f'Parameter {parameter}')
        plt.xlabel('# iteration')
        plt.ylabel('Parameter value')
        plt.grid()

        figfile = BytesIO()
        plt.savefig(figfile, format='png')
        figfile.seek(0)
        figdata_png = base64.b64encode(figfile.getvalue()).decode()
        charts.append(' ' + figdata_png)
    plt.close('all')
    return charts
```

optimizers.py

```
from itertools import product

import numpy as np
import nevergrad as ng


def n_dim_inf_generator(n=1):
    """Iterates in n-dimensional box from 0 to 1 in each dimension.

    Perform sort of exhaustive search with steps decreasing as power of 2"""
    for idx in product(*[[0, 1]]*n):
        yield np.array(idx)

    power = 1
    while True:
        k = 2**power
        one_d_vals = np.linspace(0, 1, k+1)
        for idxs in product(*[range(k+1)]*n):
            if all(not i % 2 for i in idxs):
                continue
            yield one_d_vals[list(idxs)]
        power += 1


class BaseOptimizer(ng.optimization.base.Optimizer):
    def __init__(self, *args, parametrization=None, **kwargs):
        super(BaseOptimizer, self).__init__(*args, parametrization=parametrization, **kwargs)
        self.grid_state = n_dim_inf_generator(n=parametrization.dimension)

    def _internal_ask(self):
        """Called every time `ask` request processed.

        Should be implemented for optimizer to work.
        Return 1-d array-like structure with shape same as parametrization.dimension
            Returned values should be standardized to [-3, 3] interval.
            Further mapping on hyperparameters space performed automatically
            (i.e. for scalar hyperparameter -3 will be mapped to lower limit, 3 to upper limit)
            Values out of [-3, 3] interval are truncated
            (may be useful to consider that as truncated z-index of normal distribution)"""
        return next(self.grid_state)*6-3


def get_optimizer(title):
    """Mapping of optimizer name to implementation"""
    custom_registry = {
        'GridSearch': BaseOptimizer,
        # New optimizers may be added here
    }
    optimizer = custom_registry.get(title, None)
    if optimizer is not None:
        return optimizer
```

```
optimizer = ng.optimizers.registry.get(title, None)
if optimizer is None:
    raise ValueError(f'unknown optimizer {title!r}')
return optimizer
```

Додаток В

Технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів

Опис програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б 13-1

Аркушів 8

АНОТАЦІЯ

Задля забезпечення безпеки даних та збереження незалежності від технологій користувача реалізовано сервіс, що розгортається на стороні користувача та надає прикладний програмний інтерфейс для взаємодії з користувацькими моделями, а також веб-інтерфейс для контролю процесу оптимізації.

Сервіс реалізовано мовою програмування Python, з вкористанням фреймворку Flask для імплементації прикладного програмного інтерфейсу. Тестові скрипти написані мовою Python у середовищі Jupyter Notebook.

Реалізована система розповсюджується через відкритий вихідний код а також у вигляді контейнерів docker. Рекомендовано використовувати для розгортання сервісу саме docker, а образи загружати з публічного реєстру Docker Hub.

У разі необхідності доповнення сервісу своїми оптимізаторами, рекомендовано робити відгалудження основного репозиторія та додавати нові оптимізатори відповідно до прикладу у модулі optimizers.py.

ЗМІСТ

СТРУКТУРА ВИХІДНОГО КОДУ

Вихідний код проекту структурно розбитий на три головні файли: app.py, hyperApp.py, optimizers.py. У файлі app.py оголошено об’єкт веб-додатку Flask, а також функції, що відповідають усім ендпоінтам додатку (як тим, що надають прикладний програмний інтерфейс, так і тим, що обробляють запити до веб-інтерфейсу). Це все оголошено у одному файлі для того, щоб не створювати завелику сітку детально структурованих файлів, що відповідають більш уособленим функціям. Так якімплементація усіх ендпоінтів та створення об’єкту-додатку займає усього 85 строк коду розділення цих записів не вважається доцільним. Файл hyperApp.py містить у собі оголошення класу HyperApp, що наслідується від стандартного класу Flask з фреймворку Flask. У цьому класі реалізована логіка пов’язана з спеціальними функціями для кожного ендпоінту, а також логіка підтримки стану сервісу і збереження інформації про експерименти. Файл optimizers.py містить у собі функцію для знаходження класу оптимізатору за назвою що передана користувачем у прикладний програмний інтерфейс, тобто виступає реєстром доступних класифікаторів. Також в цьому файлі присутня імплементація демонстраційного класу GridSearchOptimizer, що може слугувати взірцем та батьківським класом якщо користувачу треба добавити у сервіс новий власний оптимізатор.

ЗАПУСК СЕРВІСУ

Сервіс призначений для запуску за допомогою системи контейнеризації docker. Звісно, його можна запустити і не у контейнері, проте це потребує додаткового налаштування середовища і може конфліктувати з іншими додатками, наприклад якщо вони працюють на тому самому порті що й сервіс. З системою docker запуск сервісу зводиться до отримання docker-образу сервісу та потім запуску цього образу. Отримати образ можна загалом двома шляхами: з публічного реєстру образів Docker Hub, де даний сервіс розміщено з тегом “alex314mart/optimum”, або зібрати самостійно з вихідного коду.

Вихідний код розміщений на сервісі GitHub у публічному репозиторії “Alex314/optimum”. Для самостійної зборки docker-образу у репозиторії, у теці source/ присутній файл Dockerfile, що містить необхідні інструкції для створення образів. Такі файли є стандартом при роботі з docker. Для створення образу з вихідного коду достатньо виконати в терміналі команду «docker build . -t alex314mart/optimum» з теки source. Наведена команда створить образ з тегом «alex314mart/optimum:latest» що еквівалентно створенню образу новішого за останній доступний у Docker Hub. Для отримання останнього доступного образу з Docker Hub можна виконати команду «docker pull alex314mart/optimum». Ця команда завантажить образ з сервісу у локальний репозиторій для подальшого використання.

Після отримання образу сервісу, розгорнути сервіс можна командою «docker run -p 8001:8001 alex314mart/optimum». З корисних додаткових можливостей цієї команди слід зазначити можливість зміну порту через який буде доступний сервіс (треба замінити одну 8001 на будь-який інший номер порту). Також корисним може бути флаг «-d», що дозволяє розгортати контейнери у detached режими, тобто без необхідності підтримувати запущений термінал.

Більш детальну інформацію по командам docker можна отримати на офіційному сайті.

Додаток Г

Технології мінімізації похибок моделей машинного навчання у багатовимірному просторі гіперпараметрів

Публікації за темою роботи

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6138_20Б

Аркушів 1

УДК 004.048

О.П. Мартиненко, О.М. Шушура
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
alexander.martynenko.ua@gmail.com

ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

В задачах управління та підтримки прийняття рішень все частіше використовуються методи машинного навчання. Ці методи зазвичай параметризовані набором гіперпараметрів, від яких залежить результативність обраного методу. Метою оптимізації гіперпараметрів є пошук набору їх значень, які дають оптимальну модель, що характеризується мінімальною похибкою[1]. Автоматичний пошук оптимальних гіперпараметрів використовує як прості методи, наприклад вичерпний пошук у дискретному просторі, або випадковий пошук за заданими розподілами гіперпараметрів, так й більш складні методи, до яких відносять, наприклад, байєсівські та споріднені методи оптимізації з використанням варіантів очікуваного критерію вдосконалення. Ці методи беруть до уваги результати оцінених наборів гіперпараметрів та статистично оцінюють можливе покращення результатів у інших точках простору пошуку [2, 3].

Існують програмні пакети, які реалізують різні спеціалізовані методи оптимізації для пошуку гіперпараметрів (auto-sklearn, hyperopt, Tune, тощо)[4]. Вони дозволяють швидко налаштовувати оптимізацію гіперпараметрів у тому ж середовищі, де ведеться розробка та прототипізація моделей. Їх недоліком є прив'язка до певних фреймворків (TensorFlow, Keras, PyTorch, та інші) та мов програмування, неможливість чи складність використання у розподілених середовищах. Існують також інформаційні технології для розподіленої оптимізації гіперпараметрів (Katib, тощо). Вони зазвичай є end-to-end системами, надають можливість складних налаштуваннь, добре масштабуються, керують наданими ресурсами. Та вони зазвичай не дозволяють динамічно змінювати вже запущений процес оптимізації і часто прив'язані до певних хмарних платформ чи технологій. Платформи, що націлені на задачі машинного навчання (Amazon SageMaker, Comet.ml, тощо), вимагають виконання розрахунків і збереження користувацьких даних на їх стороні, що стає перешкодою для підприємств з підвищеними вимогами до безпеки даних[5].

Таким чином, актуальною задачею є розробка сервісу, який буде: надавати API для створення експериментів (з певним простором пошуку); зберігати результати оцінки певних наборів гіперпараметрів; проводити обчислення, пов'язані безпосередньо з пошуком потенційно оптимальних точок у просторі гіперпарамертів; надавати точки для наступних перевірок за запитом; надавати можливість редагувати налаштування експерименту між ітераціями. Такий сервіс має бути незалежним від фреймворку, мови програмування, платформи моделей машинного навчання, що оптимізуються, мати можливість розгортання на стороні користувача.

Розроблена інформаційна технологія надасть користувачеві можливість більш гнучкого контролю над обчислювальними ресурсами, що витрачаються на тренування моделей, включаючи динамічну зміну використання ресурсів під час оптимізації.

ЛІТЕРАТУРА:

1. Claesen, M. and B. D. Moor. Hyperparameter search in machine learning. arXiv 1502.02127. 2015.
2. James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*. 2012. №13(1). С. 281–305.
3. Snoek, Jasper; Larochelle, Hugo; Adams, Ryan. Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems*. arXiv:1206.2944. 2012.
4. James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms. *Proceedings of the 12th Python in Science Conference*. 2013. С. 13–20.
5. Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools. Medium. 2018. URL: https: towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a (дата звернення 22.03.20).